Introduction to clarithmetic I

Giorgi Japaridze

Abstract

"*Clarithmetic*" is a generic name for formal number theories similar to Peano arithmetic, but based on *computability logic* instead of the more traditional classical or intuitionistic logics. Formulas of clarithmetical theories represent interactive computational problems, and their "truth" is understood as existence of an algorithmic solution. Imposing various complexity constraints on such solutions yields various versions of clarithmetic. The present paper introduces a system of clarithmetic for polynomial time computability, which is shown to be sound and complete. Sound in the sense that every theorem T of the system represents an interactive number-theoretic computational problem with a polynomial time solution and, furthermore, such a solution can be efficiently extracted from a proof of T. And complete in the sense that every interactive number-theoretic problem with a polynomial time solution is represented by some theorem T of the system. The paper is written in a semitutorial style and targets readers with no prior familiarity with computability logic.

MSC: primary: 03F50; secondary: 03F30; 03D75; 03D15; 68Q10; 68T27; 68T30

Keywords: Computability logic; Interactive computation; Implicit computational complexity; Game semantics; Peano arithmetic; Bounded arithmetic

1 Introduction

Computability logic (CoL), introduced in [13, 17, 26], is a semantical, mathematical and philosophical platform, and a long-term program, for redeveloping logic as a formal theory of computability, as opposed to the formal theory of truth which logic has more traditionally been. Under the approach of CoL, formulas represent computational problems, and their "truth" is seen as algorithmic solvability. In turn, computational problems — understood in their most general, *interactive* sense — are defined as games played by a machine against its environment, with "algorithmic solvability" meaning existence of a machine that wins the game against any possible behavior of the environment. And an open-ended collection of the most basic and natural operations on computational problems forms the logical vocabulary of the theory. With this semantics, CoL provides a systematic answer to the fundamental question "*what can be computed?*", just as classical logic is a systematic tool for telling what is true. Furthermore, as it turns out, in positive cases "*what* can be computed" always allows itself to be replaced by "*how* can be computed", which makes CoL of potential interest in not only theoretical computer science, but many applied areas as well, including interactive knowledge base systems, resource oriented systems for planning and action, or declarative programming languages.

While potential applications have been repeatedly pointed out in earlier papers on CoL, so far all technical efforts had been mainly focused on finding axiomatizations for various fragments of this semantically conceived and inordinately expressive logic. Considerable advances have already been made in this direction ([14]-[16], [18]-[29], [34]), and more results in the same style are probably still to come. It should be however remembered that the main value of CoL, or anything else claiming to be a "Logic" with a capital "L", will eventually be determined by whether and how it relates to the outside, extra-logical world. In this respect, unlike many other systems officially classified as "logics", the merits of classical logic are obvious, most eloquently demonstrated by the fact that applied formal theories, a model example of which is *Peano arithmetic* **PA**, can be and have been successfully based on it. Unlike pure logics with their meaningless symbols, such theories are direct tools for studying and navigating the real world with its non-man-made, meaningful objects, such as natural numbers in the case of arithmetic. To make this point more clear to a computer scientist, one could compare a pure logic with a programming language, and applied theories based

on it with application programs written in that language. A programming language created for its own sake, mathematically or esthetically appealing but otherwise unusable as a general-purpose, comprehensive basis for application programs, would hardly be of much interest.

So, in parallel with studying possible axiomatizations and various metaproperties of pure CoL, it would certainly be worthwhile to devote some efforts to justifying its right on existence through revealing its power and appeal as a basis for applied systems. First and so far the only concrete steps in this direction have been made very recently in [27], where a CoL-based system **CLA1** of (Peano) arithmetic was constructed.[1] Unlike its classical-logic-based counterpart **PA**, **CLA1** is not merely about what arithmetical facts are *true*, but about what arithmetical problems can be actually *computed* or effectively *solved*. More precisely, every formula of the language of **CLA1** expresses a number-theoretic computational *problem* (rather than just a true/false *fact*), every theorem expresses a problem that has an algorithmic solution, and every proof encodes such a solution. Does not this sound exactly like what the constructivists have been calling for?

Unlike the mathematical or philosophical constructivism, however, and even unlike the early-day theory of computation, modern computer science has long understood that, what really matters, is not just *computability*, but rather *efficient computability*. So, the next natural step on the road of revealing the importance of CoL for computer science would be showing that it can be used for studying efficient computability just as successfully as for studying computability-in-principle. Anyone familiar with the earlier work on CoL could have found reasons for optimistic expectations here. Namely, every provable formula of any of the known sound axiomatizations of CoL happens to be a scheme of not only "always computable" problems, but "always efficiently computable" problems just as well, whatever efficiency exactly means in the context of interactive computation that CoL operates in. That is, at the level of pure logic, computability and efficient computability yield the same classes of valid principles. The study of logic abounds with phenomena in this style. One example would be the well known fact about classical logic, according to which validity with respect to all possible models is equivalent to validity with respect to just models with countable domains.

At the level of reasonably expressive applied theories, however, one should certainly expect significant differences depending on whether the underlying concept of interest is efficient computability or computability-in-principle. For instance, the earlier-mentioned system **CLA1** proves formulas expressing computable but often intractable arithmetical problems. A purpose of the present paper is to construct a CoL-based system for arithmetic which, unlike **CLA1**, proves only efficiently — specifically, polynomial time — computable problems. The new applied formal theory **CLA4** presented in Section 11 achieves this purpose. It is also a good starting point for exploring the wider class of systems under the generic name "*clarithmetic*" — arithmetical theories based on CoL, with **CLA4** being a model example of complexity-oriented versions of clarithmetic, a series of other variations of which, such as systems for polynomial space computability, primitive recursive computability, **PA**-provably recursive computability and so on, are still to come in the near future (see [30, 31]). Among the main purposes of the present piece of writing is to introduce the promising world of clarithmetic to a relatively wide audience. This explains the semitutorial style in which the paper is written. It targets readers with no prior familiarity with CoL.

Just like **CLA1**, our present system **CLA4** is not only a cognitive, but also a problem-solving tool: in order to find a solution for a given problem, it would be sufficient to write the problem in the language of the system, and find a proof of it. An algorithmic solution for the problem then would automatically come together with such a proof. However, unlike the solutions extracted from **CLA1**-proofs, which might be intractable, the solutions extracted from **CLA4**-proofs would always be efficient.

Furthermore, **CLA4** turns out to be not only sound, but also complete in a certain reasonable sense that we call *extensional completeness*. According to the latter, every number-theoretic computational problem that has a polynomial time solution is represented by some theorem of **CLA4**. Taking into account that there are many ways to represent the same problem, extensional completeness is weaker than what can be called *intensional completeness*, according to which any formula representing an (efficiently) computable problem is provable. In these terms, Gödel's celebrated theorem, here with "truth"="computability", is about intensional rather than extensional incompleteness. In fact, extensional completeness is not at all interesting in the context of classical-logic-based theories such as **PA**. In such theories, unlike CoL-based theories, it is trivially achieved, as the provable formula $\top$ represents every true sentence.

Syntactically, our **CLA4** is an extension of **PA**, and the semantics of the former is a conservative generalization of the semantics of the latter. Namely, the formulas of **PA**, which form only a proper subclass

[1] The paper [36] (in Chinese) is apparently another exception, focused on applications of CoL in AI.

of the formulas of **CLA4**, are seen as special, "moveless" sorts of problems/games, automatically solved/won when true and failed/lost when false. This makes the classical concept of truth just a special case of computability in our sense — it is nothing but computability restricted to (the problems represented by) the traditional sorts of formulas. And this means that Gödel's incompleteness theorems automatically extend from **PA** to **CLA4**, so that, unlike extensional completeness, intensional completeness in **CLA4** or any other sufficiently expressive sound CoL-based applied theory is impossible to achieve in principle. As for **CLA1**, it turns out to be incomplete in both senses. Section 15 shows that any recursively axiomatizable, sufficiently expressive, sound system would be (not only intensionally but also) extensionally incomplete, as long as the semantics of the system is based on unrestricted (as opposed to, say, efficient) computability.

Among the main moral merits of the present investigation and its contributions to the overall CoL project is an illustration of the fact that, in constructing CoL-based applied theories, successfully switching from computability to efficient computability is possible and even more than just possible. As noted, efficient computability, in fact, turns out to be much better behaved than computability-in-principle: the former allows us to achieve completeness in a sense in which the latter yields inherent incompleteness.

An advanced reader will easily understand that the present paper, while focused on the system **CLA4** of (cl)arithmetic, in fact is not only about arithmetic, but also just as much about CoL-based applied theories or knowledge base systems in general, with **CLA4** only serving as a model example of such systems. Generally, the nonlogical axioms or the knowledge base of a CoL-based applied system would be any collection of (formulas expressing) problems whose algorithmic or efficient solutions are known. Sometimes, together with nonlogical axioms, we may also have nonlogical rules of inference, preserving the property of computability or efficient computability. Then, the soundness of the corresponding underlying axiomatization of CoL (in our present case, it is system **CL12** studied in [27, 29]) — which usually comes in the strong form called *uniform-constructive soundness* — guarantees that every theorem T of the theory also has an effective or efficient solution and that, furthermore, such a solution can be effectively or efficiently extracted from a proof of T. It is this fact that, as mentioned, makes CoL-based systems problem-solving tools.

More specifically, efficiency-oriented systems in the above style and **CLA4** in particular can be seen as programming languages, where "programming" simply means theorem-proving. The soundness of the underlying system guarantees that any proof that can be written will be translatable into a program that runs efficiently and indeed is a solution of the problem expressed by the target formula of the proof. Note that the problem of verifying whether a program meets its specification, which is generally undecidable, is fully neutralized here: the "specification" is nothing but the target formula of the proof, and the proof itself, while encoding an efficient program, also automatically serves as a verification of the correctness of that program. Furthermore, every step/formula of the proof can be viewed as its own (best possible) "comment". In a more ambitious and, at this point, somewhat fantastic perspective, after developing reasonable theorem-provers, CoL-based efficiency-oriented systems can be seen as declarative programming languages in an extreme sense, where human "programming" just means writing a formula expressing the problem whose efficient solution is sought for systematic usage in the future. That is, a program simply coincides with its specification. The compiler's job would be finding a proof (the hard part) and translating it into a machine-language code (the easy part). The process of compiling could thus take long but, once compiled, the program would run fast ever after.

Various complexity-oriented systems have been studied in the literature ([2, 3, 6, 7, 8, 9, 10, 33, 35] and more). A notable advantage of CoL-based complexity-oriented systems over the other systems with similar aspirations, which typically happen to be inherently weak systems, is having actually or potentially unlimited strength, with the latter including the full expressive and deductive power of classical logic and Peano arithmetic. In view of the above-outlined potential applications, the importance of this feature is obvious: the stronger a system, the better the chances that a proof/program will be found for a declarative, non-preprocessed, ad hoc specification of the goal. Among the other appealing features of clarithmetic is being semantically meaningful in the full generality of its language, scalable, and easy to understand in its own right. Syntactically it also tends to be remarkably simple. For instance, on top of the standard Peano axioms, our present system **CLA4** only has two additional axioms $\sqcap\!x\sqcup\!y(y{=}x{+}1)$ and $\sqcap\!x\sqcup\!y(y{=}2x)$, one saying that the function $x{+}1$ is (efficiently) computable, and the other saying the same about the function $2x$. As will be seen later, from these two innocuous-looking axioms and one (also very simple) rule of induction called **CLA4**-*Induction*, via CoL, one can obtain "practically full" information about polynomial time computability of number-theoretic problems, in the same sense as **PA**, despite Gödel's incompleteness,

allows us to obtain "practically full" information about arithmetical truth. To put it in other words, if a formula F is not provable in **CLA4**, it is unlikely that anyone would find a polynomial time algorithm solving the problem expressed by F: either such an algorithm does not exist, or (as will be seen from Theorem 16.2) showing its correctness requires going beyond ordinary combinatorial reasoning formalizable in **PA**.

The closest ancestor of our present system **CLA4** is Buss's *bounded arithmetic* for polynomial time. The similarity is related to the single yet important fact that the above-mentioned rule of **CLA4**-Induction is nothing but an adaptation of Buss's PIND ("Polynomial Induction") principle to the new semantical environment in which **CLA4** operates. In this sense, **CLA4** can be characterized as a "CoL-based bounded arithmetic", as opposed to Buss's original versions of bounded arithmetic that are based on classical ([6]) or intuitionistic ([7]) logics. The switch to CoL as the logical basis for such theories creates notable differences. Among the advantages offered by this switch is absolute flexibility (as long as certain minimum-strength requirements are satisfied) in selecting the underlying "purely arithmetical" axioms. Choosing the latter to be the kind old axioms of Peano, as done in **CLA4**, allows us to achieve dramatically greater (than in the case of Buss's systems) intensional strength. Furthermore, as shown in Section 16, replacing Peano axioms with stronger ones can take us arbitrarily close to intensional completeness. This is just as far as one can go in similar pursuits because, as already noted, in view of Gödel's incompleteness phenomenon, no particular recursively enumerable system can be intensionally complete. In contrast, even "slightly" increasing the strength of the underlying (carefully hand-picked and intensionally very weak) set of arithmetical axioms in classical-logic-based or intuitionistic-logic-based bounded arithmetic immediately results in loss of soundness. We will come back to this topic in Section 17.

2 An informal overview of the main operations on games

Introducing and justifying CoL in full generality is not among the goals of the present paper — this job has been done in [13, 17, 26]. We will reintroduce only as much of (the otherwise much wider) CoL as technically necessary for understanding the system **CLA4** based on it.

As noted, formulas in CoL represent computational problems. Such problems are understood as games between two players: $\top$, called **Machine**, and $\bot$, called **Environment** (these names will not always be capitalized, and may take articles "a" or "the"). Machine is a mechanical device with fully determined, algorithmic behavior. On the other hand, there are no restrictions on the behavior of Environment. A given machine is considered to be *solving* a given problem iff it wins the corresponding game no matter how the environment acts.

Standard atomic sentences, such as "0=0" or "Peggy is John's mother", are understood as special sorts of games, called **elementary**. There are no moves in elementary games, and they are automatically won or lost. Specifically, the elementary game represented by a true sentence is won (without making any moves) by Machine, and the elementary game represented by a false sentence is won by Environment.

Logical operators are understood as operations on games/problems. One of the important groups of such operations, termed **choice operations**, comprises $\sqcap, \sqcup, \bigsqcap, \bigsqcup$. These are called **choice conjunction**, **choice disjunction**, **choice universal quantifier** and **choice existential quantifier**, respectively. $A_0 \sqcap A_1$ is a game where the first legal move ("choice"), which should be either 0 or 1, is by $\bot$. After such a move/choice i is made, the play continues and the winner is determined according to the rules of A_i; if a choice is never made, $\bot$ loses. $A_0 \sqcup A_1$ is defined in a symmetric way with the roles of $\bot$ and $\top$ interchanged: here it is $\top$ who makes an initial choice and who loses if such a choice is not made. With the universe of discourse being $\{0, 1, 10, 11, 100, \ldots\}$ (natural numbers identified with their binary representations), the meanings of the quantifiers $\bigsqcap$ and $\bigsqcup$ can now be explained by

$$\bigsqcap x A(x) = A(0) \sqcap A(1) \sqcap A(10) \sqcap A(11) \sqcap A(100) \sqcap \ldots$$

and

$$\bigsqcup x A(x) = A(0) \sqcup A(1) \sqcup A(10) \sqcup A(11) \sqcup A(100) \sqcup \ldots.$$

So, for example,

$$\bigsqcap x\big(\mathit{Prime}(x) \sqcup \mathit{Composite}(x)\big)$$

is a game where the first move is by Environment. Such a move should consist in selecting a particular number n for x, intuitively amounting to asking whether n is prime or composite. This move brings the

game down to (in the sense that the game continues as)

$$Prime(n) \sqcup Composite(n).$$

Now Machine has to move, or else it loses. The move should consist in choosing one of the two disjuncts. Let us say the left disjunct is chosen, which further brings the game down to $Prime(n)$. The latter is an elementary game, and here the interaction ends. Machine wins iff it has chosen a true disjunct. The choice of the left disjunct by Machine thus amounts to claiming/answering that n is prime. Overall, as we see, $\sqcap x\big(Prime(x) \sqcup Composite(x)\big)$ represents the problem of deciding the primality question.[2]

Similarly,

$$\sqcap x \sqcap y \sqcup z(z = x \times y)$$

is the problem of computing the product of any two numbers. Here the first two moves are by Environment, which selects some particular $m = x$ and $n = y$, thus asking Machine to tell what the product of m and n is. Machine wins if and only if, in response, it selects a (the) number k for z such that $k = m \times n$.

Another group of game operations dealt with in this paper comprises $\neg$, $\wedge$, $\vee$, $\rightarrow$. Employing the classical symbols for these operations is no accident, as they are conservative generalizations of the corresponding Boolean operations from elementary games to all games.

Negation $\neg$ is a role-switch operation: it turns $\top$'s moves and wins into $\bot$'s moves and wins, and vice versa. Since elementary games have no moves, only the winners are switched there, so that, as noted, $\neg$ acts just as the ordinary classical negation when applied to such games. For instance, as $\top$ is the winner in $0+1=1$, the winner in $\neg 0+1=1$ will be $\bot$. That is, $\top$ wins the negation $\neg A$ of an elementary game A iff it loses A, i.e., if A is false. As for the meaning of negation when applied to nonelementary games, at this point it may be useful to observe that $\neg$ interacts with choice operations in the kind old DeMorgan fashion. For example, it would not be hard to see that

$$\neg \sqcap x \sqcap y \sqcup z(z = x \times y) \;=\; \sqcup x \sqcup y \sqcap z(z \neq x \times y).$$

The operations $\wedge$ and $\vee$ are called **parallel conjunction** and **parallel disjunction**, respectively. Playing $A_0 \wedge A_1$ (resp. $A_0 \vee A_1$) means playing the two games in parallel where, in order to win, $\top$ needs to win in both (resp. at least one) of the components A_i. It is obvious that, just as in the case of negation, $\wedge$ and $\vee$ act as classical conjunction and disjunction when applied to elementary games. For instance, $0+1=1 \vee 0 \times 1 = 1$ is a game automatically won by Machine. There are no moves in it as there are no moves in either disjunct, and Machine is an automatic winner because it is so in the left disjunct. To appreciate the difference between the two — choice and parallel — groups of connectives, compare

$$\sqcap x\big(Prime(x) \sqcup \neg Prime(x)\big)$$

and

$$\sqcap x\big(Prime(x) \vee \neg Prime(x)\big).$$

The former is a computationally nontrivial problem, existence of an easy (polynomial time) solution for which had remained an open question until a few years ago. As for the latter, it is trivial, as Machine has nothing to do in it: the first (and only) move is by Environment, consisting in choosing a number n for x. Whatever n is chosen, Machine wins, as $Prime(n) \vee \neg Prime(n)$ is a true sentence and hence an automatically $\top$-won elementary game.

The operation $\rightarrow$, called **strict reduction**, is defined by $A \rightarrow B = (\neg A) \vee B$. Intuitively, this is indeed the problem of *reducing* B to A: solving $A \rightarrow B$ means solving B while having A as an external *computational resource*. Resources are symmetric to problems: what is a problem to solve for one player is a resource that the other player can use, and vice versa. Since A is negated in $(\neg A) \vee B$ and negation means switching the roles, A appears as a resource rather than problem for $\top$ in $A \rightarrow B$.

Consider $\sqcap x \sqcup y(y = x^2)$. Anyone who knows the definition of x^2 in terms of $\times$ (but perhaps does not know the meaning of multiplication, or is unable to compute this function for whatever reason) would be able to solve the problem

$$\sqcap z \sqcap u \sqcup v(v = z \times u) \;\rightarrow\; \sqcap x \sqcup y(y = x^2), \tag{1}$$

[2]For simplicity, here we treat "Composite" as the complement of "Prime", even though, strictly speaking, this is not quite so: the numbers 0 and 1 are neither prime nor composite. Writing "Nonprime" instead of "Composite" would easily correct this minor inaccuracy.

i.e., the problem

$$\sqcup z \sqcup u \sqcap v(v \neq z \times u) \ \vee \ \sqcap x \sqcup y(y = x^2),$$

as it is about reducing the consequent to the antecedent. A solution here goes like this. Wait till Environment specifies a value n for x, i.e. asks "what is the square of n?". Do not try to immediately answer this question, but rather specify the same value n for both z and u, thus asking the counterquestion: "what is n times n?". Environment will have to provide a correct answer m to this counterquestion (i.e., specify v as m where $m = n \times n$), or else it loses. Then, specify y as m, and rest your case. Note that, in this solution, Machine did not have to compute multiplication, doing which had become Environment's responsibility. Machine only correctly reduced the problem of computing square to the problem of computing product, which made it the winner.

Another group of operations that play an important role in CoL comprises $\forall$ and its dual $\exists$ (with $\exists x A(x) = \neg\forall x \neg A(x)$), called **blind universal quantifier** and **blind existential quantifier**, respectively. $\forall x A(x)$ can be thought of as a "version" of $\sqcap x A(x)$ where the particular value of x that Environment selects is invisible to Machine, so that it has to play blindly in a way that guarantees success no matter what that value is.

Compare the problems $\sqcap x\big(\mathit{Even}(x) \sqcup \mathit{Odd}(x)\big)$ and $\forall x\big(\mathit{Even}(x) \sqcup \mathit{Odd}(x)\big)$. Both of them are about telling whether a given number is even or odd; the difference is only in whether that "given number" is known to Machine or not. The first problem is an easy-to-win, two-move-deep game of a structure that we have already seen. The second game, on the other hand, is one-move deep with only Machine to make a move — select the "true" disjunct, which is hardly possible to do as the value of x remains unspecified.

Just like all other operations for which we use classical symbols, the meanings of $\forall$ and $\exists$ are exactly classical when applied to elementary games. Having this full collection of classical operations makes CoL a generalization and conservative extension of classical logic.

Going back to an earlier example, even though (1) expresses a "very easily solvable" problem, that formula is still not logically valid. Note that the success of the reduction strategy of the consequent to the antecedent that we provided for it relies on the nonlogical fact that $x^2 = x \times x$. That strategy would fail in a general case where the meanings of x^2 and $x \times x$ may not necessarily be the same. On the other hand, the goal of CoL as a general-purpose problem-solving tool should be to allow us find purely logical solutions, i.e., solutions that do not require any special, domain-specific knowledge and (thus) would be good no matter what the particular predicate or function symbols of the formulas mean. Any knowledge that might be relevant should be explicitly stated and included either in the antecedent of a given formula or in the set of axioms ("implicit antecedents" for every potential formula) of a CoL-based theory. In our present case, formula (1) easily turns into a logically valid one by adding, to its antecedent, the definition of square in terms of multiplication:

$$\forall w(w^2 = w \times w) \wedge \sqcap z \sqcap u \sqcup v(v = z \times u) \ \rightarrow \ \sqcap x \sqcup y(y = x^2). \tag{2}$$

The strategy that we provided earlier for (1) is just as good for (2), with the difference that it is successful for (2) no matter what x^2 and $z \times u$ mean, whereas, in the case of (1), it was guaranteed to be successful only under the standard arithmetical interpretations of the square and product functions. Thus, our strategy for (2) is, in fact, a "purely logical" solution.

The above examples should not suggest that blind quantifiers are meaningful or useful only when applied to elementary problems. The following is an example of a winnable nonelementary $\forall$-game:

$$\forall y\Big(\mathit{Even}(y) \sqcup \mathit{Odd}(y) \ \rightarrow \ \sqcap x\big(\mathit{Even}(x+y) \sqcup \mathit{Odd}(x+y)\big)\Big). \tag{3}$$

Solving this problem, which means reducing the consequent to the antecedent without knowing the value of y, is easy: $\top$ waits till $\bot$ selects a value n for x, and also tells — by selecting a $\sqcup$-disjunct in the antecedent — whether y is even or odd. Then, if n and y are both even or both odd, $\top$ chooses the left $\sqcup$-disjunct in the consequent, otherwise it chooses the right $\sqcup$-disjunct. Replacing the $\forall y$ prefix by $\sqcap y$ would significantly weaken the problem, obligating Environment to specify a value for y. Our strategy does not really need to know the exact value of y, as it only exploits the information about y's being even or odd, provided by the antecedent of the formula.

Many more — natural, meaningful and useful — operations beyond the ones discussed in this section have been introduced and studied within the framework of CoL. Here we have only surveyed those that are relevant to our present investigation.

3 Constant games

Now we are getting down to formal definitions of the concepts informally explained in the previous section.

To define games formally, we need certain technical terms and conventions. Let us agree that a **move** means any finite string over the standard keyboard alphabet. A **labeled move** (**labmove**) is a move prefixed with $\top$ or $\bot$, with such a prefix (**label**) indicating which player has made the move. A **run** is a (finite or infinite) sequence of labmoves, and a **position** is a finite run.

We will be exclusively using the letters Γ, Δ, Φ for runs, and α, β for moves. The letter $\wp$ will always be a variable for players, and

$$\overline{\wp}$$

will mean "$\wp$'s adversary" ("the other player"). Runs will be often delimited by "$\langle$" and "$\rangle$", with $\langle\rangle$ thus denoting the **empty run**. The meaning of an expression such as $\langle \Phi, \wp\alpha, \Gamma\rangle$ must be clear: this is the result of appending to the position $\langle\Phi\rangle$ the labmove $\langle\wp\alpha\rangle$ and then the run $\langle\Gamma\rangle$.

The following is a formal definition of what we call constant games, combined with some less formal conventions regarding the usage of certain terminology.

Definition 3.1 A **constant game** is a pair $A = (\mathbf{Lr}^A, \mathbf{Wn}^A)$, where:

1. $\mathbf{Lr}^A$ is a set of runs satisfying the condition that a (finite or infinite) run is in $\mathbf{Lr}^A$ iff all of its nonempty finite initial segments are in $\mathbf{Lr}^A$ (notice that this implies $\langle\rangle \in \mathbf{Lr}^A$). The elements of $\mathbf{Lr}^A$ are said to be **legal runs** of A, and all other runs are said to be **illegal**. We say that α is a **legal move** for $\wp$ in a position Φ of A iff $\langle \Phi, \wp\alpha\rangle \in \mathbf{Lr}^A$; otherwise α is **illegal**. When the last move of the shortest illegal initial segment of Γ is $\wp$-labeled, we say that Γ is a **$\wp$-illegal** run of A.

2. $\mathbf{Wn}^A$ is a function that sends every run Γ to one of the players $\top$ or $\bot$, satisfying the condition that if Γ is a $\wp$-illegal run of A, then $\mathbf{Wn}^A\langle\Gamma\rangle = \overline{\wp}$. When $\mathbf{Wn}^A\langle\Gamma\rangle = \wp$, we say that Γ is a **$\wp$-won** (or **won by** $\wp$) run of A; otherwise Γ is **lost** by $\wp$. Thus, an illegal run is always lost by the player who has made the first illegal move in it.

An important operation not explicitly mentioned in Section 2 is what is called *prefixation*. This operation takes two arguments: a constant game A and a position Φ that must be a legal position of A (otherwise the operation is undefined), and returns the game $\langle\Phi\rangle A$. Intuitively, $\langle\Phi\rangle A$ is the game playing which means playing A starting (continuing) from position Φ. That is, $\langle\Phi\rangle A$ is the game to which A **evolves** (will be "**brought down**") after the moves of Φ have been made. We have already used this intuition when explaining the meaning of choice operations in Section 2: we said that after $\bot$ makes an initial move $i \in \{0, 1\}$, the game $A_0 \sqcap A_1$ continues as A_i. What this meant was nothing but that $\langle\bot i\rangle(A_0 \sqcap A_1) = A_i$. Similarly, $\langle\top i\rangle(A_0 \sqcup A_1) = A_i$. Here is a definition of prefixation:

Definition 3.2 Let A be a constant game and Φ a legal position of A. The game $\langle\Phi\rangle A$ is defined by:

- $\mathbf{Lr}^{\langle\Phi\rangle A} = \{\Gamma \mid \langle\Phi, \Gamma\rangle \in \mathbf{Lr}^A\}$;
- $\mathbf{Wn}^{\langle\Phi\rangle A}\langle\Gamma\rangle = \mathbf{Wn}^A\langle\Phi, \Gamma\rangle$.

A terminological convention important to remember is that we often identify a legal position Φ of a game A with the game $\langle\Phi\rangle A$. So, for instance, we may say that the move 1 by $\bot$ brings the game $B_0 \sqcap B_1$ down to the position B_1. Strictly speaking, B_1 is not a position but a game, and what *is* a position is $\langle\bot 1\rangle$, which we here identified with the game $B_1 = \langle\bot 1\rangle(B_0 \sqcap B_1)$.

We say that a constant game A is **finite-depth** iff there is an integer d such that no legal run of A contains more than d labmoves. The smallest of such integers d is called the **depth** of A. "**Elementary**" means "of depth 0".

This paper will exclusively deal with finite-depth games. This restriction of focus makes many definitions and proofs simpler. Namely, in order to define a finite-depth-preserving game operation $O(A_1, \ldots, A_n)$ applied to such games, it suffices to specify the following:

(i) Who wins $O(A_1, \ldots, A_n)$ if no moves are made, i.e., the value of $\mathbf{Wn}^{O(A_1,\ldots,A_n)}\langle\rangle$.

(ii) What are the **initial legal (lab)moves**, i.e., the elements of $\{\wp\alpha \mid \langle\wp\alpha\rangle \in \mathbf{Lr}^{O(A_1,\ldots,A_n)}\}$, and to what game is the game $O(A_1,\ldots,A_n)$ brought down after such an initial legal labmove $\wp\alpha$ is made. Recall that, by saying that a given labmove $\wp\alpha$ brings a given game A down to B, we mean that $\langle\wp\alpha\rangle A = B$.

Then, the set of legal runs of $O(A_1,\ldots,A_n)$ will be uniquely defined, and so will be the winner in every legal (and hence finite) run of the game.

Below we define a number of operations for finite-depth games only. Each of these operations can be easily seen to preserve the finite-depth property. Of course, more general definitions of these operations — not restricted to finite-depth games — do exist (see, e.g., [26]), but in this paper we are trying to keep things as simple as possible.

Definition 3.3 Let A, B, $A_0, A_1, \ldots$ be finite-depth constant games, and n be a positive integer.

1. $\neg A$ is defined by:

   (i) $\mathbf{Wn}^{\neg A}\langle\rangle = \wp$ iff $\mathbf{Wn}^{A}\langle\rangle = \overline{\wp}$.

   (ii) $\langle\wp\alpha\rangle \in \mathbf{Lr}^{\neg A}$ iff $\langle\overline{\wp}\alpha\rangle \in \mathbf{Lr}^{A}$. Such an initial legal labmove $\wp\alpha$ brings the game down to $\neg\langle\overline{\wp}\alpha\rangle A$.

2. $A_0 \sqcap \ldots \sqcap A_n$ is defined by:

   (i) $\mathbf{Wn}^{A_0 \sqcap \ldots \sqcap A_n}\langle\rangle = \top$.

   (ii) $\langle\wp\alpha\rangle \in \mathbf{Lr}^{A_0 \sqcap \ldots \sqcap A_n}$ iff $\wp = \bot$ and $\alpha = i \in \{0,\ldots,n\}$.[3] Such an initial legal labmove $\bot i$ brings the game down to A_i.

3. $A_0 \wedge \ldots \wedge A_n$ is defined by:

   (i) $\mathbf{Wn}^{A_0 \wedge \ldots \wedge A_n}\langle\rangle = \top$ iff, for each $i \in \{0,\ldots,n\}$, $\mathbf{Wn}^{A_i}\langle\rangle = \top$.

   (ii) $\langle\wp\alpha\rangle \in \mathbf{Lr}^{A_0 \wedge \ldots \wedge A_n}$ iff $\alpha = i.\beta$, where $i \in \{0,\ldots,n\}$ and $\langle\wp\beta\rangle \in \mathbf{Lr}^{A_i}$. Such an initial legal labmove $\wp i.\beta$ brings the game down to

$$A_0 \wedge \ldots \wedge A_{i-1} \wedge \langle\wp\beta\rangle A_i \wedge A_{i+1} \wedge \ldots \wedge A_n.$$

4. $A_0 \sqcup \ldots \sqcup A_n$ and $A_0 \vee \ldots \vee A_n$ are defined exactly as $A_0 \sqcap \ldots \sqcap A_n$ and $A_0 \wedge \ldots \wedge A_n$, respectively, only with "$\top$" and "$\bot$" interchanged.

5. The infinite $\sqcap$-conjunction $A_0 \sqcap A_1 \sqcap \ldots$ is defined exactly as $A_0 \sqcap \ldots \sqcap A_n$, only with "$i \in \{0,1,\ldots\}$" instead of "$i \in \{0,\ldots,n\}$". Similarly for the infinite version of $\sqcup$.

6. In addition to the earlier-established meanings, the symbols $\top$ and $\bot$ also denote two special — simplest — constant games, defined by $\mathbf{Wn}^{\top}\langle\rangle = \top$, $\mathbf{Wn}^{\bot}\langle\rangle = \bot$ and $\mathbf{Lr}^{\top} = \mathbf{Lr}^{\bot} = \{\langle\rangle\}$.

7. $A \to B$ is treated as an abbreviation of $(\neg A) \vee B$.

Example 3.4 The game $(0{=}0 \sqcap 0{=}1) \to (10{=}11 \sqcap 10{=}10)$, i.e. $\neg(0{=}0 \sqcap 0{=}1) \vee (10{=}11 \sqcap 10{=}10)$, has thirteen legal runs, which are:

1 $\langle\rangle$. It is won by $\top$, because $\top$ is the winner in the right $\vee$-disjunct (consequent).

2 $\langle\top 0.0\rangle$. (The labmove of) this run brings the game down to $\neg 0{=}0 \vee (10{=}11 \sqcap 10{=}10)$, and $\top$ is the winner for the same reason as in the previous case.

3 $\langle\top 0.1\rangle$. It brings the game down to $\neg 0{=}1 \vee (10{=}11 \sqcap 10{=}10)$, and $\top$ is the winner because it wins in both $\vee$-disjuncts.

4 $\langle\bot 1.0\rangle$. It brings the game down to $\neg(0{=}0 \sqcap 0{=}1) \vee 10{=}11$. $\top$ loses as it loses in both $\vee$-disjuncts.

[3] Here the *number* i is identified with the standard bit *string* representing it in the binary notation. The same applies to the other clauses of this definition.

5 $\langle \bot 1.1 \rangle$. It brings the game down to $\neg(0{=}0 \sqcap 0{=}1) \vee 10{=}10$. $\top$ wins as it wins in the right $\vee$-disjunct.

6-7 $\langle \top 0.0, \bot 1.0 \rangle$ and $\langle \bot 1.0, \top 0.0 \rangle$. Both bring the game down to the false $\neg 0{=}0 \vee 10{=}11$, and both are lost by $\top$.

8-9 $\langle \top 0.1, \bot 1.0 \rangle$ and $\langle \bot 1.0, \top 0.1 \rangle$. Both bring the game down to the true $\neg 0{=}1 \vee 10{=}11$, which makes $\top$ the winner.

10-11 $\langle \top 0.0, \bot 1.1 \rangle$ and $\langle \bot 1.1, \top 0.0 \rangle$. Both bring the game down to the true $\neg 0{=}0 \vee 10{=}10$, so $\top$ wins.

12-13 $\langle \top 0.1, \bot 1.1 \rangle$ and $\langle \bot 1.1, \top 0.1 \rangle$. Both bring the game down to the true $\neg 0{=}1 \vee 10{=}10$, so $\top$ wins.

4 Games as generalized predicates

Constant games can be seen as generalized propositions: while propositions in classical logic are just elements of $\{\top, \bot\}$, constant games are functions from runs to $\{\top, \bot\}$. As we know, however, propositions only offer very limited expressive power, and classical logic needs to consider the more general concept of predicates, with propositions being nothing but special — constant — cases of predicates. The situation in CoL is similar. Our concept of a (simply) game generalizes that of a constant game in the same sense as the classical concept of a predicate generalizes that of a proposition.

We fix an infinite set of expressions called **variables**, for which we will be using $x, y, z, s, r, t, u, v, w$ as metavariables. An expression like $\vec{x}$ will usually stand for a finite sequence of variables. Similarly for later-defined objects such as constants or terms.

We also fix another infinite set of expressions called **constants**:

$$\{\epsilon, 1, 10, 11, 100, 101, 110, 111, 1000, \ldots\}.$$

These are thus **binary numerals** — the strings matching the regular expression $\epsilon \cup 1(0 \cup 1)^*$, where ϵ is the empty string. We will be typically identifying such strings — by some rather innocent abuse of concepts — with the natural numbers represented by them in the standard binary notation, and vice versa. Note that ϵ represents 0. For this reason, following the many-century tradition, we shall usually write 0 instead of ϵ, keeping in mind that, in such contexts, the length $|0|$ of the string 0 should be seen to be 0 rather than 1. We will be mostly using a, b, c, d as metavariables for constants.

A **universe** (of discourse) is a pair $(U, {}^U)$, where U is a nonempty set, and ^{U}, called the **naming function** of the universe, is a function that sends each constant c to an element c^U of U. The intuitive meaning of $c^U = \mathfrak{s}$ is that c is a **name** of $\mathfrak{s}$. Both terminologically and notationally, we will typically identify each universe $(U, {}^U)$ with its first component and, instead of "$(U, {}^U)$", write simply "U", keeping in mind that each such "universe" U comes with a fixed associated function ^{U}. A universe $U = (U, {}^U)$ is said to be **ideal** iff U coincides with the above-fixed set of constants, and ^{U} is the identity function on that set. Note that, in a non-ideal universe, such as the set of all real numbers, some objects may have several names (e.g., $1/3, 2/6, 3/9$ are different names of the same number), some have unique names (e.g. the famous number π), and some have no names at all. The same applies to the universe of astronomy, where some stars and planets have unique names, some have several names (Venus = Morning Star = Evening Star), and most have no names at all. On the other hand, the standard universe of arithmetic is ideal: every natural number has a unique name — the corresponding binary numeral — with which it can be identified.

By a **valuation** on a universe U, or a U-valuation, we mean a mapping e that sends each variable x to an element $e(x)$ of U. When a universe U is fixed, irrelevant or clear from the context, we may omit references to it and simply say "valuation". In these terms, a classical predicate p can be understood as a function that sends each valuation e to a proposition, i.e., to a constant predicate. Similarly, what we call a game sends valuations to constant games:

Definition 4.1 Let U be a universe. A **game on** U is a function A from U-valuations to constant games. We write $e[A]$ (rather than $A(e)$) to denote the constant game returned by A on valuation e. Such a constant game $e[A]$ is said to be an **instance** of A. For readability, we usually write $\mathbf{Lr}^A_e$ and $\mathbf{Wn}^A_e$ instead of $\mathbf{Lr}^{e[A]}$ and $\mathbf{Wn}^{e[A]}$.

Just as this is the case with propositions versus predicates, constant games in the sense of Definition 3.1 will be thought of as special, constant cases of games in the sense of Definition 4.1. In particular, each constant game A' is the game A such that, for every valuation e, $e[A] = A'$. From now on we will no longer distinguish between such A and A', so that, if A is a constant game, it is its own instance, with $A = e[A]$ for every e.

Where n is a natural number, we say that a game A is **n-ary** iff there are n variables such that, for any two valuations e_1 and e_2 that agree on all those variables, we have $e_1[A] = e_2[A]$. Generally, a game that is n-ary for some n, is said to be **finitary**. Our paper is going to exclusively deal with finitary games and, for this reason, we agree that, from now on, when we say "game", we usually mean "finitary game".

For a variable x and valuations e_1, e_2, we write $e_1 \equiv_x e_2$ to mean that the two valuations have the same universe and agree on all variables other than x.

We say that a game A **depends** on a variable x iff there are two valuations e_1, e_2 with $e_1 \equiv_x e_2$ such that $e_1[A] \neq e_2[A]$. An n-ary game thus depends on at most n variables. And constant games are nothing but 0-ary games, i.e., games that do not depend on any variables.

We say that a (not necessarily constant) game A is **elementary** iff so are all of its instances $e[A]$. And we say that A is **finite-depth** iff there is a (smallest) integer d, called the **depth** of A, such that the depth of no instance of A exceeds d.

Just as constant games are generalized propositions, games can be treated as generalized predicates. Namely, we will see each predicate p of whatever arity as the same-arity elementary game such that, for every valuation e, $\mathbf{Wn}^p_e\langle\rangle = \top$ iff p is true at e. And vice versa: every elementary game p will be seen as the same-arity predicate which is true at a given valuation e iff $\mathbf{Wn}^p_e\langle\rangle = \top$. Thus, for us, "predicate" and "elementary game" are going to be synonyms. Accordingly, any standard terminological or notational conventions familiar from the literature for predicates also apply to them seen as elementary games.

Just as the Boolean operations straightforwardly extend from propositions to all predicates, our operations $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\sqcap$, $\sqcup$ extend from constant games to all games. This is done by simply stipulating that $e[\ldots]$ commutes with all of those operations: $\neg A$ is the game such that, for every valuation e, $e[\neg A] = \neg e[A]$; $A \sqcap B$ is the game such that, for every valuation e, $e[A \sqcap B] = e[A] \sqcap e[B]$; etc. So does the operation of prefixation: provided that Φ is a legal position of every instance of A, $\langle\Phi\rangle A$ is understood as the unique game such that, for every valuation e, $e[\langle\Phi\rangle A] = \langle\Phi\rangle e[A]$.

Definition 4.2 Let A be a finite-depth game on a universe U, $x_1, \ldots, x_n$ be pairwise distinct variables, and $c_1, \ldots, c_n$ be constants. On the same universe, the game which we call the result of **substituting** $x_1, \ldots, x_n$ **by** $c_1, \ldots, c_n$ **in** A, denoted $A(x_1/c_1, \ldots, x_n/c_n)$, is defined by stipulating that, for every valuation e on U, $e[A(x_1/c_1, \ldots, x_n/c_n)] = e'[A]$, where e' is the valuation on U that sends each x_i to c_i and agrees with e on all other variables.

Following the standard readability-improving practice established in the literature for predicates, we will often fix pairwise distinct variables $x_1, \ldots, x_n$ for a game A and write A as $A(x_1, \ldots, x_n)$. Representing A in this form sets a context in which we can write $A(c_1, \ldots, c_n)$ to mean the same as the more clumsy expression $A(x_1/c_1, \ldots, x_n/c_n)$.

Definition 4.3 Let $A(x)$ be a finite-depth game on a given universe. On the same universe, $\sqcap xA(x)$ and $\sqcup xA(x)$ are defined as the following two games, respectively:

$$A(0) \sqcap A(1) \sqcap A(10) \sqcap A(11) \sqcap A(100) \sqcap \ldots;$$
$$A(0) \sqcup A(1) \sqcup A(10) \sqcup A(11) \sqcup A(100) \sqcup \ldots.$$

Thus, every initial legal move of $\sqcap xA(x)$ or $\sqcup xA(x)$ is a constant $c \in \{0, 1, 10, 11, 100, \ldots\}$, which in our informal language we may refer to as "the constant chosen (by the corresponding player) for x".

We will say that a game A is **unistructural** iff, for any two valuations e_1 and e_2, $\mathbf{Lr}^A_{e_1} = \mathbf{Lr}^A_{e_2}$. Of course, all constant or elementary games are unistructural. It can also be easily seen that all our game operations preserve the unistructural property of games. For the purposes of the present paper, considering only unistructural games would be sufficient.

We define the remaining operations $\forall$ and $\exists$ only for unistructural games:

Definition 4.4 Below $A(x)$ is an arbitrary finite-depth unistructural game on a universe U. On the same universe:

1. The game $\forall xA(x)$ is defined by stipulating that, for every U-valuation e, player $\wp$ and move α, we have:

   (i) $\mathbf{Wn}_e^{\forall xA(x)}\langle\rangle = \top$ iff, for every valuation g with $g \equiv_x e$, $\mathbf{Wn}_g^{A(x)}\langle\rangle = \top$.

   (ii) $\langle\wp\alpha\rangle \in \mathbf{Lr}_e^{\forall xA(x)}$ iff $\langle\wp\alpha\rangle \in \mathbf{Lr}_e^{A(x)}$. Such an initial legal labmove $\wp\alpha$ brings the game $e[\forall xA(x)]$ down to $e[\forall x\langle\wp\alpha\rangle A(x)]$.

2. The game $\exists xA(x)$ is defined in exactly the same way, only with $\top$ and $\bot$ interchanged.

Example 4.5 Consider the game (3) on the ideal universe, discussed earlier in Section 2. The sequence $\langle\bot 1.11,\ \bot 0.0,\ \top 1.1\rangle$ is a legal run of (3), the effects of the moves of which are shown below:

$$\begin{array}{ll}
(3): & \forall y\Big(\mathit{Even}(y)\sqcup\mathit{Odd}(y)\to\sqcap x\big(\mathit{Even}(x{+}y)\sqcup\mathit{Odd}(x{+}y)\big)\Big)\\
\langle\bot 1.11\rangle(3): & \forall y\big(\mathit{Even}(y)\sqcup\mathit{Odd}(y)\to\mathit{Even}(11{+}y)\sqcup\mathit{Odd}(11{+}y)\big)\\
\langle\bot 1.11,\bot 0.0\rangle(3): & \forall y\big(\mathit{Even}(y)\to\mathit{Even}(11{+}y)\sqcup\mathit{Odd}(11{+}y)\big)\\
\langle\bot 1.11,\bot 0.0,\top 1.1\rangle(3): & \forall y\big(\mathit{Even}(y)\to\mathit{Odd}(11{+}y)\big)
\end{array}$$

The play hits (ends as) the true proposition $\forall y\big(\mathit{Even}(y)\to\mathit{Odd}(11{+}y)\big)$ and hence is won by $\top$.

Before closing this section, we want to make the rather straightforward observation that the DeMorgan dualities hold for all of our sorts of conjunctions, disjunctions and quantifiers, and so does the double negation principle. That is, we always have:

$$\begin{array}{rr}
\multicolumn{2}{c}{\neg\neg A = A;}\\
\neg(A\wedge B)=\neg A\vee\neg B; & \neg(A\vee B)=\neg A\wedge\neg B;\\
\neg(A\sqcap B)=\neg A\sqcup\neg B; & \neg(A\sqcup B)=\neg A\sqcap\neg B;\\
\neg\forall xA(x)=\exists x\neg A(x); & \neg\exists xA(x)=\forall x\neg A(x);\\
\neg\sqcap xA(x)=\sqcup x\neg A(x); & \neg\sqcup xA(x)=\sqcap x\neg A(x).
\end{array}$$

5 Algorithmic strategies through interactive machines

In traditional game-semantical approaches, including Blass's [4, 5] approach which is the closest precursor of ours, player's strategies are understood as *functions* — typically as functions from interaction histories (positions) to moves, or sometimes ([1]) as functions that only look at the latest move of the history. This *strategies-as-functions* approach, however, is inapplicable in the context of CoL, whose relaxed semantics, in striving to get rid of "bureaucratic pollutants" and only deal with the remaining true essence of games, does not impose any regulations on which player can or should move in a given situation. Here, in many cases, either player may have (legal) moves, and then it is unclear whether the next move should be the one prescribed by $\top$'s strategy function or the one prescribed by the strategy function of $\bot$. For a game semantics whose ambition is to provide a comprehensive, natural and direct tool for modeling interaction, the strategies-as-functions approach would be less than adequate, even if technically possible. This is so for the simple reason that the strategies that real computers follow are not functions. If the strategy of your personal computer was a function from the history of interaction with you, then its performance would keep noticeably worsening due to the need to read the continuously lengthening — and, in fact, practically infinite — interaction history every time before responding. Fully ignoring that history and looking only at your latest keystroke in the spirit of [1] is also not what your computer does, either.

In CoL, ($\top$'s effective) strategies are defined in terms of interactive machines, where computation is one continuous process interspersed with — and influenced by — multiple "input" (environment's moves) and "output" (machine's moves) events. Of several, seemingly rather different yet equivalent, machine models of interactive computation studied in CoL, here we will employ the most basic, **HPM** ("Hard-Play Machine") model.

An HPM is nothing but a Turing machine with the additional capability of making moves. The adversary can also move at any time, with such moves being the only nondeterministic events from the machine's

perspective. Along with the ordinary work tape, the machine has an additional tape called the run tape. The latter, serving as a dynamic input, at any time spells the "current position" of the play. Its role is to make the run fully visible to the machine.

In these terms, an algorithmic solution ($\top$'s winning strategy) for a given constant game A is understood as an HPM $\mathcal{M}$ such that, no matter how the environment acts during its interaction with $\mathcal{M}$ (what moves it makes and when), the run incrementally spelled on the run tape is a $\top$-won run of A. As for $\bot$'s strategies, there is no need to define them: all possible behaviors by $\bot$ are accounted for by the different possible nondeterministic updates of the run tape of an HPM.

In the above outline, we described HPMs in a relaxed fashion, without being specific about technical details such as, say, how, exactly, moves are made by the machine, how many moves either player can make at once, what happens if both players attempt to move "simultaneously", etc. As it turns out, all reasonable design choices yield the same class of winnable games as long as we consider a certain natural subclass of games called **static**. Such games are obtained by imposing a certain simple formal condition on games (see, e.g., Section 5 of [26]), which we do not reproduce here as nothing in this paper relies on it. We shall only point out that, intuitively, static games are interactive tasks where the relative speeds of the players are irrelevant, as it never hurts a player to postpone making moves. In other words, static games are games that are contests of intellect rather than contests of speed. And one of the theses that CoL philosophically relies on is that static games present an adequate formal counterpart of our intuitive concept of "pure", speed-independent interactive computational problems. Correspondingly, CoL restricts its attention (more specifically, possible interpretations of the atoms of its formal language) to static games. All elementary games turn out to be trivially static, and the class of static games turns out to be closed under all game operations studied in CoL. More specifically, all games expressible in the language of the later-defined logic **CL12**, or theory **CLA4**, are static, as well as constant, finitary and finite-depth. Accordingly, we agree that, in this paper, we shall use the term "**computational problem**", or simply "**problem**", as a synonym of "constant, finitary, finite-depth, static game".

6 The HPM model in greater detail

As noted, computability of static games is rather robust with respect to the technical details of the underlying model of interaction. And the loose description of HPMs that we gave in the previous section would be sufficient for most purposes, just as mankind had been rather comfortably studying and using algorithms long before the Church-Turing thesis in its precise form came around. Namely, relying on just the intuitive concept of algorithmic strategies (believed in CoL to be adequately captured by the HPM model) would be sufficient if we only needed to show existence of such strategies for various games. As it happens, however, later sections of this paper need to arithmetize such strategies in order to prove the promised extensional completeness of **CLA4**. The complexity-theoretic concepts defined in the next section also require certain more specific details about HPMs, and in this section we provide such details. It should be pointed out again that most — if not all — of such details are "negotiable", as different reasonable arrangements would yield equivalent models.

Just like an ordinary Turing machine, an HPM has a finite set of **states**, one of which has the special status of being the **start state**. There are no accept, reject, or halt states, but there are specially designated states called **move states**. It is assumed that the start state is not among the move states. As noted earlier, this is a two-tape machine, with a read-write **work tape** and read-only **run tape**. Each tape has a beginning but no end, and is divided into infinitely many **cells**, arranged in the left-to-right order: cell #0, cell #1, cell #2, etc. At any time, each cell will contain one symbol from a certain fixed finite set of **tape symbols**. The **blank** symbol, as well as $\top$ and $\bot$, are among the tape symbols. We also assume that these three symbols are not among the symbols that any (legal or illegal) move can ever contain. Either tape has its own **scanning head**, at any given time looking (located) at one of the cells of the tape. A transition from one **computation step** ("**clock cycle**") to another happens according to the fixed **transition function** of the machine. The latter, depending on the current state, and the symbols seen by the two heads on the corresponding tapes, deterministically prescribes the next state, the tape symbol by which the old symbol should be overwritten in the current cell (the cell currently scanned by the head) of the work tape, and, for each head, the direction — one cell left or one cell right — in which the head should move. A constraint here is that the blank symbol, $\top$ or $\bot$ can never be written by the machine on the work tape. An attempt

to move left when the head of a given tape is looking at the leftmost cell results in staying put. So does an attempt to move right when the head is looking at the blank symbol.

When the machine starts working, it is in its start state, both scanning heads are looking at the leftmost cells of the corresponding tapes, and (all cells of) both tapes are blank (i.e., contain the blank symbol). Whenever the machine enters a move state, the string α spelled by (the contents of) its work tape cells, starting from cell #0 and ending with the cell immediately left to the work-tape scanning head, will be automatically appended — at the beginning of the next clock cycle — to the contents of the run tape in the $\top$-prefixed form $\top\alpha$. And, on every transition, whether the machine is in a move state or not, any finite sequence $\bot\beta_1, \ldots, \bot\beta_m$ of $\bot$-labeled moves may be nondeterministically appended to the contents of the run tape. If the above two events happen on the same clock cycle, then the moves will be appended to the contents of the run tape in the following order: $\top\alpha\bot\beta_1 \ldots \bot\beta_m$ (note the technicality that labmoves are listed on the run tape without blanks or commas between them).

With each labmove that emerges on the run tape, we associate its **timestamp**, which is the number of the clock cycle immediately preceding the cycle on which the move first emerged on the run tape. Intuitively, the timestamp indicates on which cycle the move was **made** rather than *appeared* on the run tape: a move made during cycle #i appears on the run tape on cycle #i+1 rather than #i. Also, we agree that the count of clock cycles, just like the count of cells, starts from 0, meaning that the very first clock cycle is cycle #0 rather than #1.

A **configuration** of a given HPM $\mathcal{M}$ is a full description of the contents of the two tapes, the locations of the two scanning heads, and the state of the machine at the beginning of some ("current") clock cycle. A **computation branch** of $\mathcal{M}$ is an infinite sequence $C_0, C_1, C_2, \ldots$ of configurations of $\mathcal{M}$, where C_0 is the initial configuration (as explained earlier), and every C_{i+1} is a configuration that could have legally followed (again, in the sense explained earlier) C_i. For a computation branch B, the **run spelled by** B is the run Γ incrementally spelled on the run tape in the corresponding scenario of interaction. We say that such a Γ is **a run generated by** the machine.

We say that a given HPM $\mathcal{M}$ **wins** (**computes**, **solves**) a given constant game A — and write $\mathcal{M} \models A$ — iff every run Γ generated by $\mathcal{M}$ is a $\top$-won run of A. We say that A is **computable** iff there is an HPM $\mathcal{M}$ such that $\mathcal{M} \models A$; such an HPM is said to be an (algorithmic) **solution**, or **winning strategy**, for A.

7 Interactive complexity

The **size** of a move α means the length of α as a string. In the context of a given computation branch of a given HPM $\mathcal{M}$, by the **background** of a clock cycle c we mean the greatest of the sizes of Environment's moves made by (before) time c, or 0 if there are no such moves. If $\mathcal{M}$ makes a move on cycle c, then the background of that move[4] means the background of c. Next, whenever $\mathcal{M}$ makes a move on cycle c, by the **timecost** of that move we mean $c - d$, where d is the greatest cycle with $d < c$ on which a move was made by either player, or is 0 if there is no such cycle.

Throughout this paper, an n-ary **arithmetical function** means a function from n-tuples of natural numbers to natural numbers. As always, "unary" means "1-ary".

Definition 7.1 Let h be a unary arithmetical function, and $\mathcal{M}$ an HPM.

1. We say that $\mathcal{M}$ **runs in time** h, or that $\mathcal{M}$ is an h **time machine**, or that h is a **bound** for the time complexity of $\mathcal{M}$, iff, in every play (computation branch), for any clock cycle c on which $\mathcal{M}$ makes a move, neither the timecost nor the size of that move exceeds $h(\ell)$, where ℓ is the background of c.

2. We say that $\mathcal{M}$ **runs in space** h, or that $\mathcal{M}$ is an h **space machine**, or that h is a **bound** for the space complexity of $\mathcal{M}$, iff, in every play (computation branch), for any clock cycle c, the number of cells ever visited by the work-tape head of $\mathcal{M}$ by time c does not exceed $h(\ell)$, where ℓ is the background of c.

Our time complexity concept can be seen to be in the spirit of what is usually called *response time*. The latter generally does not and should not depend on the length of the preceding interaction history. On the other hand, it is not and should not be merely a function of the adversary's last move, either. A similar characterization applies to our concept of space complexity. Both complexity measures are equally

[4] As easily understood, here and in similar contexts, "move" means a move not as a *string*, but as an *event*, namely, the event of $\mathcal{M}$ making a move at time c.

meaningful whether it be in the context of "short-lasting" games (such as the ones represented by the formulas of the later-defined logic **CL12**) or the context of games that may have "very long" and even infinitely long legal runs.

Let A be a constant game, h a unary arithmetical function, and $\mathcal{M}$ an HPM. We say that $\mathcal{M}$ **wins** (**computes**, **solves**) A **in time** h, or that $\mathcal{M}$ **is an** h **time solution for** A, iff $\mathcal{M}$ is an h time machine with $\mathcal{M} \models A$. We say that A is **computable** (**winnable**, **solvable**) **in time** h iff it has an h time solution. Similarly for "**space**" instead of "time".

When we say **polynomial time**, it is to be understood as "time h for some polynomial function h". Similarly for **polynomial space**.

Many concepts introduced within the framework of CoL are generalizations — for the interactive context — of ordinary and well-studied concepts of the traditional theory of computation. The above-defined time and space complexities are among such concepts. Let us focus on polynomial time for the rest of this section, and look at the traditional notion of *polynomial time computability* of a function $f(x)$ for instance. With a moment's thought, it can be seen to be equivalent to polynomial time computability (in our sense) of the problem $\sqcap x \sqcup y\big(y = f(x)\big)$. Similarly, *polynomial time decidability* of a predicate $p(x)$ means the same as polynomial time computability of the problem $\sqcap x\big(\neg p(x) \sqcup p(x)\big)$. Further, what is traditionally called (mapping) *polynomial time reducibility* of a predicate $p(x)$ to a predicate $q(x)$ can be seen to mean nothing but polynomial time computability of the problem $\sqcap x \sqcup y\big(p(x) \leftrightarrow q(y)\big)$, where $E \leftrightarrow F$ is an abbreviation of $(E \to F) \wedge (F \to E)$. If we want to say that a particular function $f(x)$ is a polynomial time reduction of $p(x)$ to $q(x)$, then we can write $\sqcap x \sqcup y\big(y = f(x)\big) \wedge \forall x\Big(p\big(x\big) \leftrightarrow q\big(f(x)\big)\Big)$. And so on.

Our formalism can be used for systematically defining and studying an infinite variety of meaningful complexity-theoretic properties, relations and operations, only some of which (as the above ones) may have established names in the literature. Consider, for instance, the problem

$$\sqcap x\big(\neg q(x) \sqcup q(x)\big) \to \sqcap x\big(\neg p(x) \sqcup p(x)\big). \tag{4}$$

It expresses (its polynomial time computability means, that is) a sort of polynomial time reducibility of $p(x)$ to $q(x)$. This reducibility can be seen to be strictly weaker than the traditional sort of polynomial time reducibility captured by the earlier mentioned $\sqcap x \sqcup y\big(y = f(x)\big)$. For instance, every predicate $p(x)$ is reducible to its complement $\neg p(x)$ in the sense of (4). Namely, a polynomial time strategy for

$$\sqcap x\big(p(x) \sqcup \neg p(x)\big) \to \sqcap x\big(\neg p(x) \sqcup p(x)\big)$$

goes as follows. Wait till a value n for x is specified in the consequent. Then specify the same value for x in the antecedent. Further wait till a $\sqcup$-disjunct is selected in the antecedent. If the first (resp. second) disjunct is selected there, select the second (resp. first) $\sqcup$-disjunct in the consequent, and celebrate victory. On the other hand, we cannot say that every predicate $p(x)$ is also polynomial time reducible — in the sense of $\sqcap x \sqcup y\big(y = f(x)\big)$ — to its complement. Take $p(x)$ to be any coNP-complete predicate. If it was polynomial time reducible to its complement, then we would have NP=coNP.

8 The language of logic CL12 and its semantics

Logic **CL12** will be axiomatically constructed in Section 9. The present section is merely devoted to its *language*. The building blocks of the formulas of the latter are:

- **Nonlogical predicate letters**, for which we use p, q as metavariables. With each predicate letter is associated a fixed nonnegative integer called its **arity**. We assume that, for any n, there are infinitely many n-ary predicate letters.
- **Function letters**, for which we use f, g as metavariables. Again, each function letter comes with a fixed **arity**, and we assume that, for any n, there are infinitely many n-ary function letters.
- The binary **logical predicate letter** $=$.
- Infinitely many **variables** and **constants**. These are the same as the ones fixed in Section 4.

Terms, for which we use $\tau, \psi, \xi, \chi, \theta, \eta$ as metavariables, are built from variables, constants and function letters in the standard way. An **atomic formula** is $p(\tau_1, \ldots, \tau_n)$, where p is an n-ary predicate letter and the τ_i are terms. When p is 0-ary, we write p instead of $p()$. Also, we write $\tau_1 = \tau_2$ instead of $=(\tau_1, \tau_2)$, and $\tau_1 \neq \tau_2$ instead of $\neg(\tau_1 = \tau_2)$. **Formulas** are built from atomic formulas, propositional connectives $\top, \bot$ (0-ary), $\neg$ (1-ary), $\wedge, \vee, \sqcap, \sqcup$ (2-ary), variables and quantifiers $\forall, \exists, \bigsqcap, \bigsqcup$ in the standard way, with the exception that, officially, $\neg$ is only allowed to be applied to atomic formulas. The definitions of *free* and *bound* occurrences of variables are also standard (with $\bigsqcap, \bigsqcup$ acting as quantifiers along with $\forall, \exists$). A formula with no free occurrences of variables is said to be **closed**.

Note that, terminologically, $\top$ and $\bot$ do not count as atoms. For us, atoms are formulas containing no logical operators. The formulas $\top$ and $\bot$ do not qualify because they *are* (0-ary) logical operators themselves.

$\neg E$, where E is not atomic, will be understood as a standard abbreviation: $\neg\top = \bot$, $\neg\neg E = E$, $\neg(A \wedge B) = \neg A \vee \neg B$, $\neg\bigsqcap x E = \bigsqcup x \neg E$, etc. And $E \to F$ will be understood as an abbreviation of $\neg E \vee F$.

Parentheses will often be omitted — as we just did — if there is no danger of ambiguity. When omitting parentheses, we assume that $\neg$ and the quantifiers have the highest precedence, and $\to$ has the lowest precedence. An expression $E_1 \wedge \ldots \wedge E_n$, where $n \geq 2$, is to be understood as $E_1 \wedge (E_2 \wedge (\ldots \wedge (E_{n-1} \wedge E_n)\ldots))$. Sometimes we can write this expression for an unspecified $n \geq 0$ (rather than $n \geq 2$). Such a formula, in the case of $n = 1$, should be understood as simply E_1. Similarly for $\vee, \sqcap, \sqcup$. As for the case of $n = 0$, $\wedge$ and $\sqcap$ should be understood as $\top$ while $\vee$ and $\sqcup$ as $\bot$.

Sometimes a formula F will be represented as $F(s_1, \ldots, s_n)$, where the s_i are variables. When doing so, we do not necessarily mean that each s_i has a free occurrence in F, or that every variable occurring free in F is among $s_1, \ldots, s_n$. However, it *will* always be assumed (usually only implicitly) that the s_i are pairwise distinct, and have no bound occurrences in F. In the context set by the above representation, $F(\tau_1, \ldots, \tau_n)$ will mean the result of replacing, in F, each occurrence of each s_i by term τ_i. When writing $F(\tau_1, \ldots, \tau_n)$, it will always be assumed (again, usually only implicitly) that the terms $\tau_1, \ldots, \tau_n$ contain no variables that have bound occurrences in F, so that there are no unpleasant collisions of variables when doing replacements.

Similar — well established in the literature — notational conventions apply to terms.

A **sequent** is an expression $E_1, \ldots, E_n \circ\!\!- F$, where $E_1, \ldots, E_n$ ($n \geq 0$) and F are formulas. Here $E_1, \ldots, E_n$ is said to be the **antecedent** of the sequent, and F said to be the **succedent**.

By a **free** (resp. **bound**) **variable** of a sequent we shall mean a variable that has a free (resp. bound) occurrence in one of the formulas of the sequent. For safety and simplicity, throughout the rest of this paper we assume that the sets of all free and bound variables of any formula or sequent that we ever consider — unless strictly implied otherwise by the context — are disjoint. This restriction, of course, does not yield any loss of expressive power, as variables can always be renamed so as to satisfy this condition.

An **interpretation** is a pair $(U, ^*)$, where $U = (U, ^U)$ is a universe and * is a function that sends:

- every n-ary function letter f to a function $f^* : U^n \to U$;
- every nonlogical n-ary predicate letter p to an n-ary predicate (elementary game) $p^*(s_1, \ldots, s_n)$ on U which does not depend on any variables other than $s_1, \ldots, s_n$.

The above uniquely extends to a mapping that sends each term τ to a function τ^*, and each formula F to a game F^*, by stipulating that:

1. $c^* = c^U$ (any constant c).
2. $s^* = s$ (any variable s).
3. Where f is an n-ary function letter and $\tau_1, \ldots, \tau_n$ are terms, $\big(f(\tau_1, \ldots, \tau_n)\big)^* = f^*(\tau_1^*, \ldots, \tau_n^*)$.
4. Where τ_1 and τ_2 are terms, $(\tau_1 = \tau_2)^*$ is $\tau_1^* = \tau_2^*$.
5. Where p is an n-ary nonlogical predicate letter and $\tau_1, \ldots, \tau_n$ are terms, $\big(p(\tau_1, \ldots, \tau_n)\big)^* = p^*(\tau_1^*, \ldots, \tau_n^*)$.
6. * commutes with all logical operators, seeing them as the corresponding game operations: $\bot^* = \bot$, $(E_1 \wedge \ldots \wedge E_n)^* = E_1^* \wedge \ldots \wedge E_n^*$, $(\bigsqcap x E)^* = \bigsqcap x (E^*)$, etc.

While an interpretation is a pair $(U, ^*)$, terminologically and notationally we will usually identify it with its second component and write * instead of $(U, ^*)$, keeping in mind that every such "interpretation" * comes

with a fixed universe U, said to be the **universe of** *. When O is a function letter, a predicate letter, a constant or a formula, and $O^* = W$, we say that * **interprets** O as W. We can also refer to such a W as "O **under interpretation** *".

When a given formula is represented as $F(x_1, \ldots, x_n)$, we will typically write $F^*(x_1, \ldots, x_n)$ instead of $\big(F(x_1, \ldots, x_n)\big)^*$. A similar practice will be used for terms as well.

We agree that, for a formula F, an interpretation * and an HPM $\mathcal{M}$, whenever we say that $\mathcal{M}$ is a **solution** of F^* or write $\mathcal{M} \models F^*$, we mean that $\mathcal{M}$ is a solution of the (constant) game $\sqcap x_1 \ldots \sqcap x_n(F^*)$, where $x_1, \ldots, x_n$ are exactly the free variables of F, listed according to their lexicographic order. We call the above game the $\sqcap$**-closure** of F^*, and denote it by $\sqcap F^*$. The $\forall$**-closure** $\forall F^*$ is defined similarly. The same notational convention extends from games to formulas.

Note that, for any given formula F, the **Lr** component of the game $\sqcap F^*$ does not depend on the interpretation *. Hence we can safely say "legal run of $\sqcap F$" without indicating an interpretation applied to the formula.

9 The axiomatization of logic CL12

Our formulations rely on some terminology and notation, explained below.

A formula not containing any choice operators $\sqcap, \sqcup, \bigsqcap, \bigsqcup$ — i.e., a formula of the language of classical first order logic — is said to be **elementary**. A sequent is **elementary** iff all of its formulas are so.

The **elementarization**

$$\|F\|$$

of a formula F is the result of replacing in F all $\sqcup$- and $\bigsqcup$-subformulas by $\bot$, and all $\sqcap$- and $\bigsqcap$-subformulas by $\top$. Note that $\|F\|$ is (indeed) an elementary formula. The **elementarization** $\|G_1, \ldots, G_n \circ\!\!- F\|$ of a sequent $G_1, \ldots, G_n \circ\!\!- F$ is the elementary formula $\|G_1\| \wedge \ldots \wedge \|G_n\| \to \|F\|$.

A sequent is said to be **stable** iff its elementarization is classically valid. By "classical validity", in view of Gödel's completeness theorem, we mean provability in some standard classical first-order calculus with constants, function letters and $=$, where $=$ is treated as the logical *identity* predicate (so that, say, $x{=}x$, $x{=}y \to (E(x) \to E(y))$, etc. are provable).

A **surface occurrence** of a subformula is an occurrence that is not in the scope of any choice operators.

We will be using the notation

$$F[E]$$

to mean a formula F together with some (single) fixed surface occurrence of a subformula E. Using this notation sets a context, in which $F[H]$ will mean the result of replacing in $F[E]$ the (fixed) occurrence of E by H. Note that here we are talking about some *occurrence* of E. Only that occurrence gets replaced when moving from $F[E]$ to $F[H]$, even if the formula also had some other occurrences of E.

By a **rule** (of inference) in this section we mean a binary relation $\mathbb{Y}\mathcal{R}X$, where $\mathbb{Y} = \langle Y_1, \ldots, Y_n\rangle$ is a finite sequence of sequents and X is a sequent. Instances of such a relation are schematically written as

$$\frac{Y_1, \ldots, Y_n}{X},$$

where $Y_1, \ldots, Y_n$ are called the **premises**, and X is called the **conclusion**. Whenever $\mathbb{Y}\mathcal{R}X$ holds, we say that X **follows** from $\mathbb{Y}$ by $\mathcal{R}$.

Expressions such as $\vec{G}, \vec{K}, \ldots$ will usually stand for finite sequences of formulas. The standard meaning of an expression such as $\vec{G}, F, \vec{K}$ should also be clear.

THE RULES OF CL12

CL12 has the six rules listed below, with the following additional conditions/explanations:

1. In $\sqcup$-Choose and $\sqcap$-Choose, $i \in \{0, 1\}$.

2. In $\sqcup$-Choose and $\sqcap$-Choose, $\mathfrak{t}$ is either a constant or a variable with no bound occurrences in the premise, and $H(\mathfrak{t})$ is the result of replacing by $\mathfrak{t}$ all free occurrences of x in $H(x)$ (rather than vice versa).

$$\textbf{$\sqcup$-Choose}\quad \frac{\vec{G} \circ\!\!- F[H_i]}{\vec{G} \circ\!\!- F[H_0 \sqcup H_1]} \qquad \textbf{$\sqcap$-Choose}\quad \frac{\vec{G},\ E[H_i],\ \vec{K} \circ\!\!- F}{\vec{G},\ E[H_0 \sqcap H_1],\ \vec{K} \circ\!\!- F}$$

$$\textbf{$\sqcup$-Choose}\quad \frac{\vec{G} \circ\!\!- F[H(\mathfrak{t})]}{\vec{G} \circ\!\!- F[\sqcup x H(x)]} \qquad \textbf{$\sqcap$-Choose}\quad \frac{\vec{G},\ E[H(\mathfrak{t})],\ \vec{K} \circ\!\!- F}{\vec{G},\ E[\sqcap x H(x)],\ \vec{K} \circ\!\!- F}$$

$$\textbf{Replicate}\quad \frac{\vec{G}, E, \vec{K}, E \circ\!\!- F}{\vec{G}, E, \vec{K} \circ\!\!- F}$$

Wait

$\dfrac{Y_1, \ldots, Y_n}{X}$ $(n \geq 0)$, where all of the following five conditions are satisfied:

1. **$\sqcap$-Condition:** Whenever X has the form $\vec{G} \circ\!\!- F[H_0 \sqcap H_1]$, both of the sequents $\vec{G} \circ\!\!- F[H_0]$ and $\vec{G} \circ\!\!- F[H_1]$ are among $Y_1, \ldots, Y_n$.
2. **$\sqcup$-Condition:** Whenever X has the form $\vec{G}, E[H_0 \sqcup H_1], \vec{K} \circ\!\!- F$, both of the sequents $\vec{G}, E[H_0], \vec{K} \circ\!\!- F$ and $\vec{G}, E[H_1], \vec{K} \circ\!\!- F$ are among $Y_1, \ldots, Y_n$.
3. **$\sqcap$-Condition:** Whenever X has the form $\vec{G} \circ\!\!- F[\sqcap x H(x)]$, for some variable y not occurring in X, the sequent $\vec{G} \circ\!\!- F[H(y)]$ is among $Y_1, \ldots, Y_n$. Here and below, $H(y)$ is the result of replacing by y all free occurrences of x in $H(x)$ (rather than vice versa).
4. **$\sqcup$-Condition:** Whenever X has the form $\vec{G}, E[\sqcup x H(x)], \vec{K} \circ\!\!- F$, for some variable y not occurring in X, the sequent $\vec{G}, E[H(y)], \vec{K} \circ\!\!- F$ is among $Y_1, \ldots, Y_n$.
5. **Stability Condition:** X is stable.

A **CL12-proof** of a sequent X is a sequence $X_1, \ldots, X_n$ of sequents, with $X_n = X$, such that, each X_i follows by one of the rules of **CL12** from some (possibly empty in the case of Wait, and certainly empty in the case of $i = 1$) set $\mathcal{P}$ of premises such that $\mathcal{P} \subseteq \{X_1, \ldots, X_{i-1}\}$. When a **CL12**-proof of X exists, we say that X is **provable** in **CL12**, and write $\mathbf{CL12} \vdash X$.

A **CL12-proof** of a formula F will be understood as a **CL12**-proof of the empty-antecedent sequent $\circ\!\!- F$. Accordingly, $\mathbf{CL12} \vdash F$ means $\mathbf{CL12} \vdash \circ\!\!- F$.

CL12 is a conservative extension of classical logic (see [27]). Namely, an elementary sequent $E_1, \ldots, E_n \circ\!\!- F$ is provable in **CL12** iff the formula $E_1 \wedge \ldots \wedge E_n \to F$ is valid in the classical sense. It is also a conservative extension of the earlier known logic **CL3** studied in [18].[5] The latter is nothing but the empty-antecedent fragment of **CL12** without function letters and identity.

[5] An essentially the same logic, under the name **L**, was in fact known as early as in [12].

Example 9.1 In this example, $\times$ is a binary function letter and ^{3} is a unary function letter. We write $x \times y$ and x^3 instead of $\times(x, y)$ and $^3(x)$, respectively. The following sequence of sequents is a **CL12**-proof (of its last sequent):

1. $\forall x\big(x^3 = (x \times x) \times x\big),\ t = s \times s,\ r = t \times s \ \circ\!\!-\ r = s^3$ Wait: (no premises)
2. $\forall x\big(x^3 = (x \times x) \times x\big),\ t = s \times s,\ r = t \times s \ \circ\!\!-\ \sqcup y(y = s^3)$ $\sqcup$-Choose: 1
3. $\forall x\big(x^3 = (x \times x) \times x\big),\ t = s \times s,\ \sqcup z(z = t \times s) \ \circ\!\!-\ \sqcup y(y = s^3)$ Wait: 2
4. $\forall x\big(x^3 = (x \times x) \times x\big),\ t = s \times s,\ \sqcap y \sqcup z(z = t \times y) \ \circ\!\!-\ \sqcup y(y = s^3)$ $\sqcap$-Choose: 3
5. $\forall x\big(x^3 = (x \times x) \times x\big),\ t = s \times s,\ \sqcap x \sqcap y \sqcup z(z = x \times y) \ \circ\!\!-\ \sqcup y(y = s^3)$ $\sqcap$-Choose: 4
6. $\forall x\big(x^3 = (x \times x) \times x\big),\ \sqcup z(z = s \times s),\ \sqcap x \sqcap y \sqcup z(z = x \times y) \ \circ\!\!-\ \sqcup y(y = s^3)$ Wait: 5
7. $\forall x\big(x^3 = (x \times x) \times x\big),\ \sqcap y \sqcup z(z = s \times y),\ \sqcap x \sqcap y \sqcup z(z = x \times y) \ \circ\!\!-\ \sqcup y(y = s^3)$ $\sqcap$-Choose: 6
8. $\forall x\big(x^3 = (x \times x) \times x\big),\ \sqcap x \sqcap y \sqcup z(z = x \times y),\ \sqcap x \sqcap y \sqcup z(z = x \times y) \ \circ\!\!-\ \sqcup y(y = s^3)$ $\sqcap$-Choose: 7
9. $\forall x\big(x^3 = (x \times x) \times x\big),\ \sqcap x \sqcap y \sqcup z(z = x \times y) \ \circ\!\!-\ \sqcup y(y = s^3)$ Replicate: 8
10. $\forall x\big(x^3 = (x \times x) \times x\big),\ \sqcap x \sqcap y \sqcup z(z = x \times y) \ \circ\!\!-\ \sqcap x \sqcup y(y = x^3)$ Wait: 9

Example 9.2 The formula $\forall x\, p(x) \to \sqcap x\, p(x)$ is provable in **CL12**. It follows from $\forall x\, p(x) \to p(y)$ by Wait. The latter, in turn, follows by Wait from the empty set of premises.

On the other hand, the formula $\sqcap x\, p(x) \to \forall x\, p(x)$, i.e. $\sqcup x \neg p(x) \vee \forall x\, p(x)$, in not provable. Indeed, its elementarization is $\bot \vee \forall x\, p(x)$, which is not classically valid. Hence $\sqcup x \neg p(x) \vee \forall x\, p(x)$ cannot be derived by Wait. Replicate can also be dismissed for obvious reasons. This leaves us with $\sqcup$-Choose. But if $\sqcup x \neg p(x) \vee \forall x\, p(x)$ is derived by $\sqcup$-Choose, then the premise should be $\neg p(\mathfrak{t}) \vee \forall x\, p(x)$ for some variable or constant $\mathfrak{t}$. The latter, however, is a classically non-valid elementary formula and hence unprovable.

Example 9.3 The formula $\sqcap x \sqcup y\big(p(x) \to p(y)\big)$ is provable in **CL12** as follows:

1. $p(s) \to p(s)$ Wait:
2. $\sqcup y\big(p(s) \to p(y)\big)$ $\sqcup$-Choose: 1
3. $\sqcap x \sqcup y\big(p(x) \to p(y)\big)$ Wait: 2

On the other hand, the formula $\sqcup y \sqcap x\big(p(x) \to p(y)\big)$ can be seen to be unprovable, even though its classical counterpart $\exists y \forall x\big(p(x) \to p(y)\big)$ is a classically valid elementary formula and hence provable in **CL12**.

Example 9.4 While the formula $\forall x \exists y\big(y = f(x)\big)$ is classically valid and hence provable in **CL12**, its constructive counterpart $\sqcap x \sqcup y\big(y = f(x)\big)$ can be easily seen to be unprovable. This is no surprise. In view of the expected soundness of **CL12**, provability of $\sqcap x \sqcup y\big(y = f(x)\big)$ would imply that every function f is computable, which, of course, is not the case.

Exercise 9.5 To see the resource-consciousness of **CL12**, show that it does not prove $p \sqcap q \to (p \sqcap q) \wedge (p \sqcap q)$, even though this formula has the form $F \to F \wedge F$ of a classical tautology. Then show that, in contrast, **CL12** proves the *sequent* $p \sqcap q \ \circ\!\!-\ (p \sqcap q) \wedge (p \sqcap q)$ because, unlike the antecedent of a $\to$-combination, the antecedent of a $\circ\!\!-$-combination is reusable (trough Replicate).

Exercise 9.6 Show that $\mathbf{CL12} \vdash \sqcup x \sqcap y\, p(x, y) \ \circ\!\!-\ \sqcup x\big(\sqcap y\, p(x, y) \wedge \sqcap y\, p(x, y)\big)$. Then observe that, on the other hand, **CL12** does not prove any of the formulas

$$\begin{aligned}
\sqcup x \sqcap y\, p(x, y) &\to \sqcup x\big(\sqcap y\, p(x, y) \wedge \sqcap y\, p(x, y)\big);\\
\sqcup x \sqcap y\, p(x, y) \ \wedge\ \sqcup x \sqcap y\, p(x, y) &\to \sqcup x\big(\sqcap y\, p(x, y) \wedge \sqcap y\, p(x, y)\big);\\
\sqcup x \sqcap y\, p(x, y) \ \wedge\ \sqcup x \sqcap y\, p(x, y) \ \wedge\ \sqcup x \sqcap y\, p(x, y) &\to \sqcup x\big(\sqcap y\, p(x, y) \wedge \sqcap y\, p(x, y)\big);\\
&\ldots
\end{aligned}$$

10 The adequacy of logic CL12

Logical Consequence (**LC**) is the following rule, with both the premises and the conclusion being formulas:

From $E_1, \ldots, E_n$ *conclude* F*, as long as* **CL12** *proves* $E_1, \ldots, E_n \circ\!\!- F$.

When F follows from $E_1, \ldots, E_n$ by this rule, i.e., when $\mathbf{CL12} \vdash E_1, \ldots, E_n \circ\!\!- F$, we say that F is a **logical consequence** of $E_1, \ldots, E_n$.

The official terms of the language of **CL12**, identified with their parse trees, are tree-style structures, so let us call them **tree-terms**. A more general and economical way to represent terms, however, is to allow merging some or all identical-content nodes in such trees, thus turning them into (directed, acyclic, rooted, edge-ordered multi-) graphs. Let us call these unofficial sorts of terms **graph-terms**. The idea of representing linguistic objects in the form of graphs rather than trees is central in the approach called *cirquent calculus* ([16, 23]), and has already proven its worth. We once again find the usefulness of that idea in our present, complexity-sensitive context. Figure 1 illustrates two terms representing the same polynomial function y^8, with the term on the right being a tree-term and the term on the left being a graph-term. As this example suggests, graph-terms are generally exponentially smaller than the corresponding tree-terms, which explains our interest in the former. Figure 1 also makes it unnecessary to formally define graph-terms, as their meaning must be perfectly clear after looking at this single example.

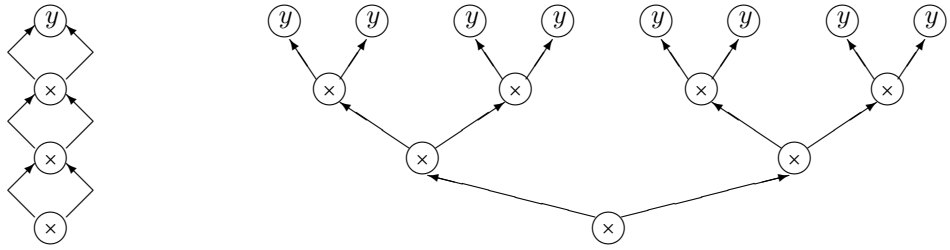

Figure 1: A graph-term and the corresponding tree-term

By a **polynomial graph-term** τ we shall mean a graph-term not containing (at its leaves) any constants other than 0, and not containing (at its internal nodes) any function letters other than $'$ (unary), $+$ (binary) and $\times$ (binary). The total number k of the variables $y_1, \ldots, y_k$ occurring in (at the leaves of) τ is said to be the **arity** of τ. Subsequently we shall only be interested in unary (1-ary) polynomial graph-terms, and will typically omit the word "unary". Terminologically and notationally we shall usually identify such a term τ with the unary polynomial arithmetical function represented by it under the standard arithmetical interpretation (x' means $x+1$). So, for instance, either term of Figure 1 is a polynomial graph-term, representing — and identified with — the function y^8.

We generalize the above concept of a (unary) polynomial graph-term τ to that of a $(1, n)$-ary **explicit polynomial functional** by allowing the term τ to contain, on top of variables and $0, ', +, \times$, additional n ($n \geq 0$) unary function letters $f_1, \ldots, f_n$, semantically treated as placeholders for unary arithmetical functions. We say that such a τ **depends on** $f_1, \ldots, f_n$. Replacing $f_1, \ldots, f_n$ by names $g_1, \ldots, g_n$ of some particular unary arithmetical functions turns τ into the corresponding unary arithmetical function, which we shall denote by $\tau(g_1, \ldots, g_n)$. For instance, the term of Figure 2 is a $(1, 2)$-ary explicit polynomial functional. Let us denote it by τ. Then, if g means "square" and h means "cube", $\tau(g, h)$ is the unary arithmetical function $(y^2 + y^3)^3$.

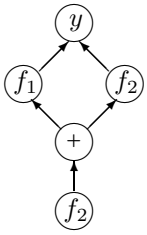

Figure 2: An explicit polynomial functional expressing $f_2\big(f_1(y)+f_2(y)\big)$

We further generalize the concept of a (unary) polynomial graph-term τ to that of a (unary) **explicit polynomial function**. The latter is defined as a nonempty sequence $\langle \tau_{f_1}, \ldots, \tau_{f_k} \rangle$ of explicit polynomial functionals indexed by (associated with) pairwise distinct unary function letters $f_1, \ldots, f_k$, where each τ_{f_i} does not depend on any function letters that are not among $f_1, \ldots, f_{i-1}$. Note that this condition makes τ_{f_1} simply a polynomial graph-term, i.e., a $(1,0)$-ary explicit polynomial functional. (Meta)semantically, each index f_i here is a name of (represents) a unary arithmetical function, and the corresponding τ_{f_i} is a definition of that function in terms of 0, $'$, $+$, $\times$ and some earlier-defined functions; then τ itself is stipulated to represent the same function as f_k does. Again, a single example would be sufficient to fully clarify the denotation of each explicit polynomial function. Consider the explicit polynomial function $\tau = \langle \tau_{f_1}, \tau_{f_1}, \tau_{f_3} \rangle$, where τ_{f_1} and τ_{f_2} are the $(1,0)$-ary explicit polynomial functionals of Figure 1, and τ_{f_3} is the $(1,2)$-ary explicit polynomial functional of Figure 2. Thus, both f_1 and f_2 represent the same unary polynomial function y^8, and f_3, i.e. τ itself, represents the unary polynomial function $(y^8+y^8)^8$. Of course, every explicit polynomial function can be translated into an equivalent polynomial graph-term, but such a translation can increase the size exponentially. For our purposes, polynomial graph-terms (let alone tree-terms) are too inefficient means of representing polynomial functions. This explains our preference for explicit polynomial functions as a standard way of writing (in our metalanguage) polynomial terms.

As in the case of polynomial graph-terms, terminologically and notationally we shall usually identify an explicit polynomial function τ with the unary arithmetical function represented by it. When τ is an explicit polynomial function and $\mathcal{M}$ is a τ time (resp. space) machine, we say that τ is an **explicit polynomial bound** for the time (resp. space) complexity of $\mathcal{M}$.

Another auxiliary concept that we are going to rely on in this section and later is that of a **generalized HPM** (**GHPM**). For a natural number n, an n-ary GHPM is defined in the same way as an HPM, with the difference that the former takes n natural numbers as inputs (say, provided on a separate, read-only *input tape*); such inputs are present at the very beginning of the work of the machine and remain unchanged throughout it. An ordinary HPM is thus nothing but a 0-ary GHPM. When $\mathcal{M}$ is an n-ary GHPM and $c_1, \ldots, c_n$ are natural numbers, $\mathcal{M}(c_1, \ldots, c_n)$ denotes the HPM that works just like $\mathcal{M}$ in the scenario where the latter has received $c_1, \ldots, c_n$ as inputs. We will assume that some reasonable encoding (through natural numbers) of GHPMs is fixed. When $\mathcal{M}$ is a GHPM, $\ulcorner \mathcal{M} \urcorner$ denotes its **code**.

The paper [29] established the soundness of **CL12** in the following strong sense:

Theorem 10.1 *([29]) If a formula F is a logical consequence of formulas $E_1, \ldots, E_n$ and * is an interpretation such that each E_i^* $(1 \leq i \leq n)$ is computable, then so is F^*. Furthermore:*

1. *There is an efficient*[6] *procedure that takes an arbitrary* **CL12***-proof of an arbitrary sequent $E_1, \ldots, E_n \circ\!\!- F$ and constructs a n-ary GHPM $\mathcal{M}$, together with a $(1,n)$-ary explicit polynomial functional τ, such that, for any interpretation *, any n-ary GHPMs $\mathcal{N}_1, \ldots, \mathcal{N}_n$ and any unary arithmetical functions $g_1, \ldots, g_n$, if each $\mathcal{N}_i(\ulcorner \mathcal{N}_1 \urcorner, \ldots, \ulcorner \mathcal{N}_n \urcorner)$ is a g_i time solution of E_i^*, then $\mathcal{M}(\ulcorner \mathcal{N}_1 \urcorner, \ldots, \ulcorner \mathcal{N}_n \urcorner)$ is a $\tau(g_1, \ldots, g_n)$ time solution of F^*.*

2. *The same holds for "space" instead of "time".*

Among the corollaries of the above theorem is that LC preserves both polynomial time and polynomial space computabilities, as well as Ω-time and Ω-space computabilities for any class Ω of functions containing

[6] Here and later in similar metacontexts, "efficient" means "polynomial time".

all polynomial functions and closed under composition (such as, say, the class of all primitive recursive functions). Indeed, to see that LC preserves (for instance) polynomial time computability, assume F is a logical consequence of $E_1, \ldots, E_n$, and the HPMs $\mathcal{N}_1, \ldots, \mathcal{N}_n$ are polynomial time solutions of $E_1, \ldots, E_n$. Of course, every such HPM $\mathcal{N}_i$ can as well be seen as an n-ary GHPM which simply ignores its inputs. So, each $\mathcal{N}_i(\ulcorner\mathcal{N}_1\urcorner, \ldots, \ulcorner\mathcal{N}_n\urcorner)$ solves E_i. Then, according to Theorem 10.1, we can construct a polynomial time solution $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner, \ldots, \ulcorner\mathcal{N}_n\urcorner)$ of F. Furthermore, such a solution, together with an explicit polynomial bound for its time complexity, can be efficiently extracted from $\mathcal{N}_1, \ldots, \mathcal{N}_n$, explicit polynomial bounds for their time complexities, and the **CL12**-proof of $E_1, \ldots, E_n \circ\!\!- F$.

Remember that, philosophically speaking, computational *resources* are symmetric to computational problems: what is a problem for one player to solve is a resource that the other player can use. Namely, having a problem A as a computational resource intuitively means having the ability to successfully solve/win A. For instance, as a resource, $\sqcap x \sqcup y(y = x^2)$ means the ability to tell the square of any number.

Together with soundness, [29] also established the completeness of **CL12**. While in the present paper we treat $\circ\!\!-$ merely as a syntactic expression separating the two parts of a sequent, in [29] it is seen as an operation on games, called **ultimate reduction**. For simplicity considerations, we do not want to reproduce the definition of $\circ\!\!-$ here, which would be necessary to precisely state the completeness result for **CL12**. We shall only point out that, intuitively, $A_1, \ldots, A_n \circ\!\!- B$ is (indeed) the problem of reducing B to $A_1, \ldots, A_n$ in the most general intuitive sense. It is similar to $A_1 \wedge \ldots \wedge A_n \to B$, but with the difference that, during playing this game, in the former $\top$ can use any of the antecedental resources A_i repeatedly, while in the latter at most once. The A_is, as resources, are "recyclable", that is. Furthermore, they are "recyclable" in the strongest sense possible. Namely, it is not necessary to restart A_i from the beginning every time it is reused. $\top$ may as well choose to continue A_i — in a new way — from any of the previously reached positions. This corresponds to the way of reusage any purely software resource would offer in the presence of an advanced operating system and unlimited memory: one can start running process A_i, then fork it at any stage thus creating two copies (branches) with a common past but possibly diverging futures, then fork any of the new branches again at any time, and so on. See [26] or [29] for more details and explanations.

Anyway, the completeness result of [29] for **CL12** says that, if there is an HPM that solves the problem $E_1^*, \ldots, E_n^* \circ\!\!- F^*$ for any interpretation * — such an *-independent HPM is said to be a *logical solution* of the sequent $E_1, \ldots, E_n \circ\!\!- F$ — then $\mathbf{CL12} \vdash E_1, \ldots, E_n \circ\!\!- F$. Furthermore, the same has been shown to hold even if HPMs are no longer required to follow algorithmic (let alone efficient) strategies — for instance, if they are allowed to use oracles for whatever functions. This result, in view of the intuitions captured by the operation $\circ\!\!-$, eventually translates in [29] into the following thesis relevant to our further purposes:

Thesis 10.2 ([29]) Assume $E_1, \ldots, E_n, F$ are formulas such that there is a *-independent (whatever interpretation *) intuitive description and justification of a winning strategy for F^*, which relies on the availability and "recyclability" — in the strongest sense possible — of $E_1^*, \ldots, E_n^*$ as computational resources. Then F is a logical consequence of $E_1, \ldots, E_n$.

The above means that the rule of Logical Consequence lives up to its name, and that we can always reliably use intuition on games and strategies when reasoning about them in **CL12**-based systems (where LC is a rule of inference) such as **CLA4**. Namely, once a formula E is proven in such a system, it can be treated as a recyclable resource whose unlimited availability can be safely assumed for any new strategies that we construct. More precisely, a new strategy may assume that there is an (external) **provider** of the resource E (call it an **oracle** for E if you prefer), capable of successfully playing E for (against) us any time and in any number of sessions, whether we choose those sessions to evolve in a parallel or a branched/forked fashion.

Example 10.3 Imagine a **CL12**-based applied formal theory, in which we have already proven two facts: $\forall x\big(x^3 = (x \times x) \times x\big)$ (the meaning of "cube" in terms of multiplication) and $\sqcap x \sqcap y \sqcup z(z = x \times y)$ (the computability of multiplication), and now we want to derive $\sqcap x \sqcup y(y = x^3)$ (the computability of "cube"). This is how we can reason to justify $\sqcap x \sqcup y(y = x^3)$:

> *Consider any s (selected by Environment for x in $\sqcap x \sqcup y(y = x^3)$). We need to find s^3. Using the resource $\sqcap x \sqcap y \sqcup z(z = x \times y)$ twice, we first find the value t of $s \times s$, and then the value r of $t \times s$. According to $\forall x(x^3 = (x \times x) \times x)$, such an r is the sought s^3.*

Thesis 10.2 promises that the above intuitive argument will be translatable into a **CL12**-proof of

$$\forall x\big(x^3=(x\times x)\times x\big),\ \sqcap x\sqcap y\sqcup z(z=x\times y)\ \circ\!\!-\ \sqcap x\sqcup y(y=x^3)$$

(and hence the succedent will be derivable in the theory by LC as the formulas of the antecedent are already proven). Such a proof indeed exists — see Example 9.1.

CL12 — more precisely, the associated rule of LC — is adequate because, on one hand, by Theorem 10.1, it is sound for a wide spectrum of applied theories, including — but not limited to — polynomial-time-oriented ones, and, on the other hand, by Thesis 10.2, it is as strong as a logical rule of inference could possibly be.

11 Theory CLA4 introduced

The language of **CLA4**, whose **formulas** more specifically can be referred to as **CLA4-formulas**, is obtained from the language of **CL12** by removing all nonlogical predicate letters (thus only leaving the logical predicate letter $=$), removing all constants but 0, and removing all but three function letters, which are:

- *successor*, unary. We will write τ' for $successor(\tau)$.
- *sum*, binary. We will write $\tau_1+\tau_2$ for $sum(\tau_1,\tau_2)$.
- *product*, binary. We will write $\tau_1\times\tau_2$ for $product(\tau_1,\tau_2)$.

Thus, the language of **CLA4** extends that of **Peano arithmetic PA** (see, for example, [11]) through adding to it $\sqcap$, $\sqcup$, $\bigsqcup$, $\bigsqcap$. Formulas that have no free occurrences of variables are said to be **sentences**.

The concept of an interpretation explained earlier can now be restricted to interpretations that are only defined on $'$, $+$ and $\times$, as the present language has no other nonlogical function or predicate letters. Of such interpretations, the **standard interpretation** † is the one whose universe is the ideal universe $\{0,1,10,11,100,\ldots\}$, with its elements identified with the corresponding natural numbers, and which interprets $'$ as the standard successor $(x+1)$ function, interprets $+$ as the sum function, and interprets $\times$ as the product function. For a **CLA4**-formula F, the **standard interpretation of** F is the game $F^\dagger$, which we typically write simply as F unless doing so may cause ambiguity.

Where τ is a term, we will be using $\tau\mathbf{0}$ and $\tau\mathbf{1}$ as abbreviations for the terms $0''\times\tau$ and $(0''\times\tau)'$, respectively. The choice of this notation is related to the fact that, given any natural number a, the binary representation of $0''\times a$ (i.e., of $2a$) is nothing but the binary representation of a with a "0" added on its right. Similarly, the binary representation of $(0''\times a)'$ is nothing but the binary representation of a with a "1" added to it. Of course, here an exception is the case $a=0$. It can be made an ordinary case by assuming that adding any number of 0s at the beginning of a binary numeral b results in a legitimate numeral representing the same number as b does.

The number $a\mathbf{0}$ (i.e. $2a$) will be said to be the **binary 0-successor** of a, and $a\mathbf{1}$ (i.e. $2a+1$) said to be the **binary 1-successor** of a; in turn, we can refer to a as the **binary predecessor** of $a\mathbf{0}$ and $a\mathbf{1}$. As for a', we can refer to it as the **unary successor** of a, and refer to a as the **unary predecessor** of a'. Every number has a binary predecessor, and every number except 0 has a unary predecessor. Note that the binary predecessor of a number is the result of deleting the last digit (if present) in its binary representation. Remember that the string 0 for us is just another name of the empty string ϵ representing the number zero. So, 0 and 1 are no exceptions to the above rule: deleting the last (and only) digit of 1 results in ϵ; and "deleting the last digit" in 0, i.e. in ϵ, again results in ϵ, as there is no digit to delete.

The language of **PA** is known to be very expressive, despite its nonlogical vocabulary officially being limited to only $0, ', +, \times$. Specifically, it allows us to express, in a certain standard way, all recursive functions and relations, and beyond. Relying on the common knowledge of the power of the language of **PA**, we will be using standard expressions such as $x\le y$, $y>x$, etc. in formulas as abbreviations of the corresponding proper expressions of the language. In our metalanguage, $|x|$ will refer to the length of the binary numeral for the number represented by x. In other words, $|x|=\lceil log_2(x+1)\rceil$ ($\lceil y\rceil$ means the smallest integer z with $y\le z$). Expressions like $|x|$ we refer to as **pseudoterms** — officially they are not terms of the object language, but in many contexts still can be treated as such. Another example of a pseudoterm

is 2^x with its standard meaning. So, when we write, say, "$|x| \leq y$", it is officially to be understood as an abbreviation of a standard formula of **PA** saying that $|x|$ does not exceed y.

For a variable x, by a **polynomial sizebound** for x we shall mean a standard formula of the language of **PA** saying that $|x| \leq \tau(|y_1|, \ldots, |y_n|)$, where $y_1, \ldots, y_n$ are any variables different from x, and $\tau(|y_1|, \ldots, |y_n|)$ is any $(0, ', +, \times)$-combination of $|y_1|, \ldots, |y_n|$. For instance, $|x| \leq |y| + |z|$ is a polynomial sizebound for x, which is a formula of **PA** saying that the size of x does not exceed the sum of the sizes of y and z. Now, we say that a **CLA4**-formula F is **polynomially bounded** iff:

- Whenever $\sqcap xG(x)$ is a subformula of F, $G(x)$ has the form $S(x) \to H(x)$, where $S(x)$ is a polynomial sizebound for x.
- Whenever $\sqcup xG(x)$ is a subformula of F, $G(x)$ has the form $S(x) \wedge H(x)$, where $S(x)$ is a polynomial sizebound for x.

Remember that, where F is a formula, $\forall F$ means the $\forall$-closure of F, i.e., $\forall x_1 \ldots \forall x_n F$, where $x_1, \ldots, x_n$ are the free variables of F. Similarly for $\exists F$, $\sqcap F$, $\sqcup F$.

The **axioms** of **CLA4** are:

Axiom 1: $\forall x(0 \neq x')$

Axiom 2: $\forall x \forall y(x' = y' \to x = y)$

Axiom 3: $\forall x(x + 0 = x)$

Axiom 4: $\forall x \forall y\big(x + y' = (x + y)'\big)$

Axiom 5: $\forall x(x \times 0 = 0)$

Axiom 6: $\forall x \forall y\big(x \times y' = (x \times y) + x\big)$

Axiom 7: $\forall\Big(F(0) \wedge \forall x\big(F(x) \to F(x')\big) \to \forall x F(x)\Big)$ for each elementary formula $F(x)$

Axiom 8: $\sqcap x \sqcup y(y = x')$

Axiom 9: $\sqcap x \sqcup y(y = x\mathbf{0})$

All of the above are thus *nonlogical axioms* (**CLA4** has no logical axioms). Note that the overall number of axioms is infinite rather than nine, because Axiom 7 is not a particular sentence but a scheme of sentences. Axioms 1-7 are nothing but **Peano axioms** — the nonlogical axioms of **PA**. **CLA4** thus only has two **extra-Peano axioms**: Axiom 8 and Axiom 9. In view of the forthcoming soundness theorem for **CLA4**, Axiom 8 says that the unary successor function is polynomial time computable, and Axiom 9 says the same about the binary 0-successor function.

As for the **rules of inference**, **CLA4** has a single logical rule, which is our old friend Logical Consequence, and a single nonlogical rule, which we call **CLA4-Induction**. The latter is

$$\frac{\sqcap\big(F(0)\big) \qquad \sqcap\big(F(x) \to F(x\mathbf{0})\big) \qquad \sqcap\big(F(x) \to F(x\mathbf{1})\big)}{\sqcap\big(F(x)\big)},$$

where $F(x)$ is any *polynomially bounded* formula.

Here we shall say that $\sqcap\big(F(0)\big)$ is the **basis** of induction, $\sqcap\big(F(x) \to F(x\mathbf{0})\big)$ is the **left inductive step**, and $\sqcap\big(F(x) \to F(x\mathbf{1})\big)$ is the **right inductive step**. The variable x has a special status here, and we say that the conclusion follows from the premises by **CLA4-Induction on** x. A reader familiar with Buss's [6] bounded arithmetic will notice a resemblance between the PIND axiom scheme of the latter and our **CLA4**-Induction rule, even though the two beasts operate in very different environments, of course.

A sentence F is considered **provable** in **CLA4** iff there is a sequence of sentences, called a **CLA4-proof** of F, where each sentence is either an axiom, or follows from some previous sentences by one of the two rules of inference, and where the last sentence is F. An **extended CLA4-proof** is defined in the same way, only, with the additional requirement that each application of LC should come together with an attached **CL12**-proof of the corresponding sequent. With some fixed, effective, sound and complete axiomatization L

of classical first order logic in mind, a **superextended CLA4-proof** is an extended **CLA4**-proof with the additional requirement that every application of Wait in the justification of a **CL12**-derivation in it comes with an L-proof of the elementarization of the conclusion. Note that the property of being a superextended proof is (efficiently) decidable, while the properties of being an extended proof or just a proof are only recursively enumerable. We write $\mathbf{CLA4} \vdash F$ to say that F is provable (has a proof) in **CLA4**, and $\mathbf{CLA4} \nvdash F$ to say the opposite.

Generally, as in the above definition of provability and proofs, in **CLA4** we will only be interested in proving *sentences*.[7] So, for technical convenience, we agree that, from now on, whenever we write $\mathbf{CLA4} \vdash F$ (or say "F is provable") for a non-sentence F, it simply means that $\mathbf{CLA4} \vdash \sqcap F$. Similarly, when we say that a given strategy solves F, it is to be understood as that the strategy solves $\sqcap F$. Similarly, when we say that F is a logical consequence of $E_1, \ldots, E_n$, what we typically mean is that $\sqcap F$ is a logical consequence of $\sqcap E_1, \ldots, \sqcap E_n$. To summarize, in the context of **CLA4**, any formula with free variables should be understood as its $\sqcap$-closure. An exception is when $F(x_1, \ldots, x_n)$ is an elementary formula and we say that $F(c_1, \ldots, c_n)$ is **true** (whatever constants $c_1, \ldots, c_n$). This is to be understood as that the $\forall$-closure $\forall F(c_1, \ldots, c_n)$ of $F(c_1, \ldots, c_n)$ is true (in the standard model), for "truth" is only meaningful for elementary formulas (which $\sqcap F$ generally would not be). An important fact on which we will often rely yet only implicitly so, is that the sentence $\forall F \to \sqcap F$ or the closed sequent $\forall F \circ\!\!- \sqcap F$ is (always) **CL12**-provable. In view of the soundness of **CL12**, this means that whenever (F is elementary and) $\forall F$ is true, $\sqcap F$ is automatically won by a strategy that does nothing.

Example 11.1 The following sequence is a proof of $\sqcap x \sqcup y(y{=}x\mathbf{1})$, i.e. of $\sqcap x \sqcup y\big(y{=}(x\mathbf{0})'\big)$ — the sentence saying that the binary 1-successor function is computable:

I. $\sqcap x \sqcup y(y{=}x')$ Axiom 8

II. $\sqcap x \sqcup y(y{=}x\mathbf{0})$ Axiom 9

III. $\sqcap x \sqcup y\big(y{=}(x\mathbf{0})'\big)$ LC: I,II

An extended version of the above proof would have to include a justification for step III where LC was used, such as the following one:

1. $t{=}r'$, $r{=}s\mathbf{0} \circ\!\!- t{=}(s\mathbf{0})'$ Wait:
2. $t{=}r'$, $r{=}s\mathbf{0} \circ\!\!- \sqcup y\big(y{=}(s\mathbf{0})'\big)$ $\sqcup$-Choose: 1
3. $\sqcup y(y{=}r')$, $r{=}s\mathbf{0} \circ\!\!- \sqcup y\big(y{=}(s\mathbf{0})'\big)$ Wait: 2
4. $\sqcap x \sqcup y(y{=}x')$, $r{=}s\mathbf{0} \circ\!\!- \sqcup y\big(y{=}(s\mathbf{0})'\big)$ $\sqcap$-Choose: 3
5. $\sqcap x \sqcup y(y{=}x')$, $\sqcup y(y{=}s\mathbf{0}) \circ\!\!- \sqcup y\big(y{=}(s\mathbf{0})'\big)$ Wait: 4
6. $\sqcap x \sqcup y(y{=}x')$, $\sqcap x \sqcup y(y{=}x\mathbf{0}) \circ\!\!- \sqcup y\big(y{=}(s\mathbf{0})'\big)$ $\sqcap$-Choose: 5
7. $\sqcap x \sqcup y(y{=}x')$, $\sqcap x \sqcup y(y{=}x\mathbf{0}) \circ\!\!- \sqcap x \sqcup y\big(y{=}(x\mathbf{0})'\big)$ Wait:

As we just saw, (additionally) justifying an application of LC takes more space than the (non-extended) proof itself. And this would be a typical case for **CLA4**-proofs. Luckily, however, there is no real need to formally justify LC. Firstly, this is so because **CL12** is an analytic system, and proof-search in it is a routine (even if sometimes long) syntactic exercise. Secondly, in view of Thesis 10.2, there is no need to generate formal **CL12**-proofs anyway: instead, we can use intuition on games and strategies. In the present case, the whole (III+7)-step extended formal proof can be replaced by a short intuitive argument in the following style:

> *Consider any s chosen by Environment for x. We need to compute $(s\mathbf{0})'$. Using Axiom 9, we find the value r of $s\mathbf{0}$. Then, using Axiom 8, we find the value t of r'. Such a t is what we are looking for.*

Furthermore, as we agreed to understand a proof of a non-sentence F as a proof of its $\sqcap$-closure $\sqcap F$, the above argument can be shortened by considering simply $\sqcup y(y{=}x\mathbf{1})$ instead of $\sqcap x \sqcup y(y{=}x\mathbf{1})$. This would allow us to skip the phrase "Consider any s chosen by Environment for x" and the necessity to introduce

[7] In case we do not insist that every formula in an **CLA4**-proof be a sentence, one could show that, if F is not a sentence, it is provable (in this relaxed sense) if and only if its $\sqcap$-closure is so, anyway.

the new name/variable s. Instead, we can simply say "Consider any x", implicitly letting x itself serve as a variable for the value chosen by Environment for x when playing the $\sqcap$-closure of the game. Sometimes we can go even lazier and skip the phrase "Consider any x" altogether.

In view of the following fact, an alternative way to present **CLA4** would be to delete Axioms 1-7 and, instead, declare all (closed) theorems of **PA** to be axioms of **CLA4** along with Axioms 8 and 9:

Fact 11.2 *Every (elementary* **CLA4***-) sentence provable in* **PA** *is also provable in* **CLA4***.*

Proof. Suppose (the classical-logic-based) **PA** proves F. By the deduction theorem for classical logic this means that, for some nonlogical axioms $H_1, \ldots, H_n$ of **PA**, the formula $H_1 \wedge \ldots \wedge H_n \rightarrow F$ is provable in classical first order logic. Hence $H_1 \wedge \ldots \wedge H_n \circ\!\!- F$ is provable in **CL12** by Wait from the empty set of premises. Hence F is a logical consequence of the Peano axioms $H_1, \ldots, H_n$ of **CLA4** and, as such, is provable by LC. ■

The above fact, on which we will be implicitly relying in the sequel, allows us to construct "lazy" **CLA4**-proofs where some steps can be justified by simply indicating their provability in **PA**. That is, we will treat theorems of **PA** as if they were axioms of **CLA4**. As **PA** is well known and well studied, we safely assume that the reader has a good feel for what it can prove, so we do not usually further justify **PA**-provability claims that we make. A reader less familiar with **PA**, can take it as a rule of thumb that, despite Gödel's incompleteness theorems, **PA** proves every true number-theoretic fact that a contemporary high school student can establish, or that mankind was or could be aware of before 1931.

Example 11.3 The following sequence is a lazy proof of $\sqcap x(x{=}0 \sqcup x{\neq}0)$ — the formula saying that the "zeroness" predicate is decidable:

I. $0{=}0 \sqcup 0{\neq}0$ LC:

II. $\forall x(x{=}0 \rightarrow x\mathbf{0}{=}0)$ **PA**

III. $\forall x(x{\neq}0 \rightarrow x\mathbf{0}{\neq}0)$ **PA**

IV. $\sqcap x(x{=}0 \sqcup x{\neq}0 \rightarrow x\mathbf{0}{=}0 \sqcup x\mathbf{0}{\neq}0)$ LC: II,III

V. $\forall x(x\mathbf{1}{\neq}0)$ **PA**

VI. $\sqcap x(x{=}0 \sqcup x{\neq}0 \rightarrow x\mathbf{1}{=}0 \sqcup x\mathbf{1}{\neq}0)$ LC: V

VII. $\sqcap x(x{=}0 \sqcup x{\neq}0)$ **CLA4**-Induction: I,IV,VI

An extended version of the above proof will include the following three additional justifications (**CL12**-proofs):

A justification for Step I:

1. $0{=}0$ Wait:
2. $0{=}0 \sqcup 0{\neq}0$ $\sqcup$-Choose: 1

A justification for Step IV:

1. $\forall x(x{=}0 \rightarrow x\mathbf{0}{=}0),\ \forall x(x{\neq}0 \rightarrow x\mathbf{0}{\neq}0) \ \circ\!\!-\ s{=}0 \rightarrow s\mathbf{0}{=}0$ Wait:
2. $\forall x(x{=}0 \rightarrow x\mathbf{0}{=}0),\ \forall x(x{\neq}0 \rightarrow x\mathbf{0}{\neq}0) \ \circ\!\!-\ s{=}0 \rightarrow s\mathbf{0}{=}0 \sqcup s\mathbf{0}{\neq}0$ $\sqcup$-Choose: 1
3. $\forall x(x{=}0 \rightarrow x\mathbf{0}{=}0),\ \forall x(x{\neq}0 \rightarrow x\mathbf{0}{\neq}0) \ \circ\!\!-\ s{\neq}0 \rightarrow s\mathbf{0}{\neq}0$ Wait:
4. $\forall x(x{=}0 \rightarrow x\mathbf{0}{=}0),\ \forall x(x{\neq}0 \rightarrow x\mathbf{0}{\neq}0) \ \circ\!\!-\ s{\neq}0 \rightarrow s\mathbf{0}{=}0 \sqcup s\mathbf{0}{\neq}0$ $\sqcup$-Choose: 3
5. $\forall x(x{=}0 \rightarrow x\mathbf{0}{=}0),\ \forall x(x{\neq}0 \rightarrow x\mathbf{0}{\neq}0) \ \circ\!\!-\ s{=}0 \sqcup s{\neq}0 \rightarrow s\mathbf{0}{=}0 \sqcup s\mathbf{0}{\neq}0$ Wait: 2,4
6. $\forall x(x{=}0 \rightarrow x\mathbf{0}{=}0),\ \forall x(x{\neq}0 \rightarrow x\mathbf{0}{\neq}0) \ \circ\!\!-\ \sqcap x(x{=}0 \sqcup s{\neq}0 \rightarrow x\mathbf{0}{=}0 \sqcup x\mathbf{0}{\neq}0)$ Wait: 5

A justification for Step VI:

1. $\forall x(x\mathbf{1}{\neq}0) \ \circ\!\!-\ s{=}0 \rightarrow s\mathbf{1}{\neq}0$ Wait:
2. $\forall x(x\mathbf{1}{\neq}0) \ \circ\!\!-\ s{\neq}0 \rightarrow s\mathbf{1}{\neq}0$ Wait:
3. $\forall x(x\mathbf{1}{\neq}0) \ \circ\!\!-\ s{=}0 \sqcup s{\neq}0 \rightarrow s\mathbf{1}{\neq}0$ Wait: 1,2

4. $\forall x(x\mathbf{1}\neq 0)$ $\circ\!\!-$ $s=0\sqcup s\neq 0\to s\mathbf{1}=0\sqcup s\mathbf{1}\neq 0$ $\sqcup$-Choose: 3

5. $\forall x(x\mathbf{1}\neq 0)$ $\circ\!\!-$ $\sqcap x(x=0\sqcup x\neq 0\to x\mathbf{1}=0\sqcup x\mathbf{1}\neq 0)$ Wait: 4

But, again, in the sequel we will not generate formal (even if lazy) proofs in the above style but, instead, limit ourselves to informal arguments within **CLA4**. Below we illustrate such an argument for the present case. Further, as explained in the previous example, we prefer to deal with $x=0\sqcup x\neq 0$ rather than $\sqcap x(x=0\sqcup x\neq 0)$, and do not explicitly say "consider any value x chosen for the variable x by Environment".

We prove $x=0\sqcup x\neq 0$ *by* **CL12***-Induction on* x*.*

The Basis $0=0\sqcup 0\neq 0$ *of induction is straightforward: it is solved by choosing the left* $\sqcup$*-disjunct.*

Solving the left inductive step $x=0\sqcup x\neq 0\to x\mathbf{0}=0\sqcup x\mathbf{0}\neq 0$ *means solving the consequent using a single copy of the antecedent as a resource. From* **PA***, we know that* $x\mathbf{0}=0$ *iff* $x=0$*. And whether* $x=0$ *or not we can find out using the resource* $x=0\sqcup x\neq 0$*. So, we can tell whether* $x\mathbf{0}=0$ *or (*$\sqcup$*)* $x\mathbf{0}\neq 0$*, as desired.*

The right inductive step $x=0\sqcup x\neq 0\to x\mathbf{1}=0\sqcup x\mathbf{1}\neq 0$ *is solved by simply choosing the right* $\sqcup$*-disjunct in the consequent which, by* **PA***, is true.*

In the future, our reliance on **PA** may not always be explicit as it was in the above informal argument. Further, as we advance, our reliance on some basic nonelementary facts such as the just-proven $\sqcap x(x=0\sqcup x\neq 0)$ may increasingly become only implicit.

Why did not we use the following, much simpler intuitive argument instead of the above one?

> *To solve* $x=0\sqcup x\neq 0$*, see if* x *(the constant chosen by Environment for* x*) is* 0 *or not. If yes, choose the left* $\sqcup$*-disjunct; otherwise choose the right* $\sqcup$*-disjunct.*

The above argument is not valid, in the sense that it is not "purely logical", and hence Thesis 10.2 does not guarantee that it will be translatable into a formal proof. Namely, in being confident that the above strategy wins the game, we rely on the extra-logical knowledge of the fact that different constants are names of different objects. This is indeed so for the standard model of arithmetic. But in some other models it is quite possible that, say, both constants 0 and 10 are names for the same object of the universe. Then, the above strategy prescribes to choose the false $10\neq 0$ in the corresponding scenario.

Thus, $\sqcap x(x=0\sqcup x\neq 0)$, while provable in **CLA4**, is not logically valid, that is, is not provable in **CL12**. The same applies to the more general principle $\sqcap x\sqcap y(x=y\sqcup x\neq y)$, whose **CLA4**-provability is shown later in Section 12.

When a formula $F(x_1,\ldots,x_n)$ is a standard (in whatever informal sense) representation of an n-ary predicate p and $\mathbf{CLA4}\vdash\sqcap\big(F(x_1,\ldots,x_n)\sqcup\neg F(x_1,\ldots,x_n)\big)$, we say that p is **CLA4-provably decidable**. Similarly, when $F(y,x_1,\ldots,x_n)$ is a standard representation of the graph of an n-ary function f and $\mathbf{CLA4}\vdash\sqcap\sqcup yF(y,x_1,\ldots,x_n)$, we say that f is **CLA4-provably computable**. Since **CLA4** is the only system of clarithmetic dealt with in this paper, the prefix "**CLA4**" before "provably" can be safely omitted. So, for instance, Example 11.3 established the provable decidability of the "zeroness" predicate, and Example 11.1 established the provable computability of the binary 1-successor function. The same terminology extends to partially defined functions as well. For instance, the later-proven Fact 12.3 shows that $\mathbf{CLA4}\vdash\sqcap x\big(x\neq 0\to\sqcup y(x=y')\big)$. We understand this as the provable computability of the (partial) unary predecessor function.

Exercise 11.4 Let $Even(x)$ be an abbreviation of $\exists z(x=z+z)$, and $Odd(x)$ be an abbreviation of $\neg Even(x)$. Find a lazy **CLA4**-proof of $\forall x\Big(Even(x)\sqcup Odd(x)\to\sqcap y\big(Even(x+y)\sqcup Odd(x+y)\big)\Big)$. *Hint*: Relying on the **PA**-provable fact that $x\mathbf{0}$ is always even and $x\mathbf{1}$ is always odd, by **CLA4**-Induction prove $\sqcap x\big(Even(x)\sqcup Odd(x)\big)$. The target sentence is a logical consequence of the latter and the **PA**-provable fact that the sum of two numbers is even iff both numbers are even or both are odd.

By an **arithmetical problem** in this paper we mean a game A such that, for some sentence F of the language of **CLA4**, $A=F^{\dagger}$ (remember that † is the standard interpretation). Such a sentence F is said a **representation** of A. We say that an arithmetical problem A is **provable** in **CLA4** iff it has a **CLA4**-provable representation. In these terms, a central result of the present paper sounds as follows:

Theorem 11.5 *An arithmetical problem has a polynomial time solution iff it is provable in* **CLA4***.*

Furthermore, there is an efficient procedure that takes an arbitrary extended **CLA4***-proof of an arbitrary sentence X and constructs a solution of X (of $X^\dagger$, that is) together with an explicit polynomial bound for its time complexity.*

Proof. The soundness ("if") part of this theorem will be proven in Section 13, and the completeness ("only if") part in Section 14. ■

12 Some more taste of CLA4

In this section we establish several **CLA4**-provability facts. In view of the soundness of **CLA4**, each such fact tells us about the efficient solvability of the associated number-theoretic computational problem.

As mentioned, the present work has been written not merely as a research paper but also as an advanced tutorial on CoL-based applied theories. The series of proofs given in this section can be treated as exercises aimed at developing the reader's feel for CoL-based systems and **CLA4** in particular. But these proofs also provide certain necessary results relied upon later, in our proof of the extensional completeness of **CLA4**.

As noted earlier, we shall exclusively rely on informal reasoning in **CLA4**, remembering that behind every such piece of reasoning is a formal **CLA4**-proof.

Fact 12.1 $\mathbf{CLA4} \vdash \sqcup y(y{=}|x|)$ *(i.e.,* $\mathbf{CLA4} \vdash \sqcap x \sqcup y(y{=}|x|)$*).*

Proof. Argue in **CLA4**. By **CLA4**-Induction on x, we first want to prove $\sqcup y(|y| \leq |x| \wedge y{=}|x|)$.

The basis $\sqcup y(|y| \leq |0| \wedge y{=}|0|)$ is solved by selecting 0 for y. Such a move brings the game down to $|0| \leq 0 \wedge 0{=}|0|$. From **PA**, we know that the latter is true. So, we win.

The left inductive step is $\sqcup y(|y| \leq |x| \wedge y{=}|x|) \to \sqcup y(|y| \leq |x\mathbf{0}| \wedge y{=}|x\mathbf{0}|)$. Using Example 11.3, we figure our whether $x{=}0$ or not. If $x{=}0$, then (we know from **PA** that) $|x\mathbf{0}| = 0$, so we win the game by selecting 0 for y in the consequent. Now suppose $x{\neq}0$. We wait till Environment selects a constant a for y in the antecedent[8] (if this does not happen, we win). Then, using Axiom 8, we compute the value (constant) b with $b{=}a'$, and choose b for y in the consequent. We win because, from **PA**, we know that when x is not 0, the resulting position $|a| \leq |x| \wedge a{=}|x| \to |b| \leq |x\mathbf{0}| \wedge b{=}|x\mathbf{0}|$, i.e. $|a| \leq |x| \wedge a{=}|x| \to |a'| \leq |x\mathbf{0}| \wedge a'{=}|x\mathbf{0}|$ (whatever the value of x is) is true. The right inductive step is similar but simpler, as we do not need to separately consider the case of $x{=}0$.

So, we know how to win (the $\sqcap$-closure of) $\sqcup y(|y| \leq |x| \wedge y{=}|x|)$. But then, of course, ignoring the $|y| \leq |x|$ conjunct, we also know how to win (the $\sqcap$-closure of the weaker) $\sqcup y(y{=}|x|)$. Putting it in precise terms, $\sqcap x \sqcup y(y{=}|x|)$ is an (easy) logical consequence of $\sqcap x \sqcup y(|y| \leq |x| \wedge y{=}|x|)$ — that is, the sequent $\sqcap x \sqcup y(|y| \leq |x| \wedge y{=}|x|) \circ\!\!- \sqcap x \sqcup y(y{=}|x|)$ is (easily) provable in **CL12**. ■

Compare the $\sqcap$-closures of the following formulas:

$$\exists y(x{=}y\mathbf{0} \vee x{=}y\mathbf{1}) \tag{5}$$

$$\sqcup y(x{=}y\mathbf{0} \vee x{=}y\mathbf{1}) \tag{6}$$

$$\exists y(x{=}y\mathbf{0} \sqcup x{=}y\mathbf{1}) \tag{7}$$

$$\sqcup y(x{=}y\mathbf{0} \sqcup x{=}y\mathbf{1}) \tag{8}$$

All four sentences "say the same" about the arbitrary ($\sqcap$) number represented by x, but in different ways. (5) is the weakest, least informative, of the four. It says that x has a binary predecessor y, and that x is even (i.e., is the binary 0-successor of its binary predecessor) or odd (i.e., is the binary 1-successor of its binary predecessor). This is an almost trivial piece of information. (6) and (7) carry stronger information. According to (6), x not just merely *has* a binary predecessor y, but such a predecessor can be actually and efficiently *found*. (7) strengthens (5) in another way. It says that x can be efficiently determined to be even

[8] In informal arguments like this, we usually do not try to be consistent in using different metavariables for constants and variables. Notice that the status of x in the present informal argument is also "constant" (chosen by Environment for the variable x) just like the status of a but we, out of reluctance to introduce new names, continue using the expression "x" for it, even though in earlier sections we tried to reserve the metanames $x, y, z, \ldots$ (as opposed to $a, b, c, \ldots$) for variables rather than constants.

or odd. As for (8), it is the strongest. It carries two pieces of good news at once: we can efficiently find the binary predecessor y of x and, simultaneously, tell whether x is even or odd. According to the following fact, (8) is provable. As we may guess, so are the weaker (7), (6), (5).

Fact 12.2 **CLA4** $\vdash \sqcup y(x=y\mathbf{0}\sqcup x=y\mathbf{1})$.

Proof. Argue in **CLA4**. By **CLA4**-Induction on x, we first want to prove $\sqcup y\big(|y|\leq|x|\wedge(x=y\mathbf{0}\sqcup x=y\mathbf{1})\big)$.

The basis $\sqcup y\big(|y|\leq|0|\wedge(0=y\mathbf{0}\sqcup 0=y\mathbf{1})\big)$ is obviously solved by selecting 0 for y and then choosing the left $\sqcup$-disjunct, which results in the true (according to **PA**) sentence $|0|\leq|0|\wedge 0=0\mathbf{0}$. The left inductive step $\sqcup y(x=y\mathbf{0}\sqcup x=y\mathbf{1})\to\sqcup y(x\mathbf{0}=y\mathbf{0}\sqcup x\mathbf{0}=y\mathbf{1})$ is solved by selecting for y the same constant as the one selected by Environment for x, and then choosing the left $\sqcup$-disjunct. Similarly, the right inductive step $\sqcup y(x=y\mathbf{0}\sqcup x=y\mathbf{1})\to\sqcup y(x\mathbf{1}=y\mathbf{0}\sqcup x\mathbf{1}=y\mathbf{1})$ is solved by selecting for y the same constant as the one selected by Environment for x, and then choosing the right $\sqcup$-disjunct.

Now, $\sqcup y(x=y\mathbf{0}\sqcup x=y\mathbf{1})$ is a straightforward logical consequence of $\sqcup y\big(|y|\leq|x|\wedge(x=y\mathbf{0}\sqcup x=y\mathbf{1})\big)$. ■

The preceding fact established the provable computability of binary predecessor. The following fact does the same for unary predecessor:

Fact 12.3 **CLA4** $\vdash x\neq 0\to\sqcup y(x=y')$.

Proof. Argue in **CLA4**. By **CLA4**-Induction on x, we want to prove $x\neq 0\to\sqcup y(|y|\leq|x|\wedge x=y')$, from which the target $x\neq 0\to\sqcup y(x=y')$ follows immediately by LC.

The basis $0\neq 0\to\sqcup y(|y|\leq|0|\wedge 0=y')$ is solved trivially by a strategy that does nothing. A win is guaranteed because the antecedent is false.

The left inductive step is $\big(x\neq 0\to\sqcup y(|y|\leq|x|\wedge x=y')\big)\to\big(x\mathbf{0}\neq 0\to\sqcup y(|y|\leq|x\mathbf{0}|\wedge x\mathbf{0}=y')\big)$. If $x\mathbf{0}\neq 0$ (and if not, we win the game), then — according to **PA** — $x\neq 0$. So, Environment will have to choose a constant a for y in the antecedent, or else it loses. We may assume that a is (indeed) the unary predecessor of x, or else, again, having chosen a wrong a, Environment loses. We know from **PA** that then the unary predecessor b of $x\mathbf{0}$ equals $a\mathbf{1}$, and that $|b|\leq|x\mathbf{0}|$. This b can be computed using (the resource provided by) Example 11.1. We choose b for y in the consequent and win.

The right inductive step $\big(x\neq 0\to\sqcup y(|y|\leq|x|\wedge x=y')\big)\to\big(x\mathbf{1}\neq 0\to\sqcup y(|y|\leq|x\mathbf{1}|\wedge x\mathbf{1}=y')\big)$ is even easier to handle. Prom **PA**, the unary predecessor of $x\mathbf{1}$ is $x\mathbf{0}$, and $|x\mathbf{0}|\leq|x\mathbf{1}|$. Using Axiom 9, we compute the value b of $x\mathbf{0}$ and choose b for y in the consequent. ■

Fact 12.4 **CLA4** $\vdash \sqcup z(z=x+y)$.

Proof. Argue in **CLA4**. By **CLA4**-Induction on x, we want to prove

$$\sqcap y\big(|y|\leq|t|\to\sqcup z(|z|\leq|x|+|y|\wedge z=x+y)\big),$$

from which, together with the **PA**-provable fact $\forall y(|y|\leq|y|)$, the target $\sqcup z(z=x+y)$ follows by LC.

The basis is $\sqcap y\big(|y|\leq|t|\to\sqcup z(|z|\leq|0|+|y|\wedge z=0+y)\big)$, which (as always) we prefer to simply write as $|y|\leq|t|\to\sqcup z(|z|\leq|0|+|y|\wedge z=0+y)$. It is won by selecting (the value of) y for z, because we know, from **PA**, that the resulting $|y|\leq|t|\to|y|\leq|0|+|y|\wedge y=0+y)$ is true.

In inductive steps, we will rely on the fact that **PA** proves (the $\forall$-closures of) the following formulas:

$$s\mathbf{0}+r\mathbf{0}=(s+r)\mathbf{0},\quad\text{i.e.,}\quad 2s+2r=2(s+r);\tag{9}$$

$$s\mathbf{0}+r\mathbf{1}=(s+r)\mathbf{1},\quad\text{i.e.,}\quad 2s+(2r+1)=2(s+r)+1;\tag{10}$$

$$s\mathbf{1}+r\mathbf{0}=(s+r)\mathbf{1},\quad\text{i.e.,}\quad (2s+1)+2r=2(s+r)+1;\tag{11}$$

$$s\mathbf{1}+r\mathbf{1}=\big((s+r)\mathbf{1}\big)',\quad\text{i.e.,}\quad (2s+1)+(2r+1)=\big(2(s+r)+1\big)+1.\tag{12}$$

The left inductive step is

$$\sqcap y\big(|y|\leq|t|\to\sqcup z(|z|\leq|x|+|y|\wedge z=x+y)\big)\to\sqcap y\big(|y|\leq|t|\to\sqcup z(|z|\leq|x\mathbf{0}|+|y|\wedge z=x\mathbf{0}+y)\big).\tag{13}$$

To solve it, we wait till Environment chooses a constant a for y in the consequent, after which (13) will be brought down to

$$\sqcap y\big(|y|\leq|t|\rightarrow\sqcup z(|z|\leq|x|+|y|\wedge z=x+y)\big)\rightarrow\big(|a|\leq|t|\rightarrow\sqcup z(|z|\leq|x\mathbf{0}|+|a|\wedge z=x\mathbf{0}+a)\big). \qquad (14)$$

Using Fact 12.2, we find the binary predecessor b of a, for which we will also know whether $a=b\mathbf{0}$ or ("or" in the strong sense of $\sqcup$) $a=b\mathbf{1}$. We specify y as b in the antecedent of (14), and wait till Environment selects a value c for z there.

If $a=b\mathbf{0}$, the game by now will be brought down to

$$\big(|b|\leq|t|\rightarrow(|c|\leq|x|+|b|\wedge c=x+b)\big)\rightarrow\big(|b\mathbf{0}|\leq|t|\rightarrow\sqcup z(|z|\leq|x\mathbf{0}|+|b\mathbf{0}|\wedge z=x\mathbf{0}+b\mathbf{0})\big).$$

Using Axiom 9, we compute the value d of $c\mathbf{0}$, and specify z as d in the consequent. The resulting position

$$\big(|b|\leq|t|\rightarrow(|c|\leq|x|+|b|\wedge c=x+b)\big)\rightarrow\big(|b\mathbf{0}|\leq|t|\rightarrow|c\mathbf{0}|\leq|x\mathbf{0}|+|b\mathbf{0}|\wedge c\mathbf{0}=x\mathbf{0}+b\mathbf{0}\big),$$

in view of (9) (and certain additional, straightforward **PA**-provable facts), is true, so we win.

Quite similarly, if $a=b\mathbf{1}$, the game by now will be brought down to

$$\big(|b|\leq|t|\rightarrow(|c|\leq|x|+|b|\wedge c=x+b)\big)\rightarrow\big(|b\mathbf{1}|\leq|t|\rightarrow\sqcup z(|z|\leq|x\mathbf{0}|+|b\mathbf{1}|\wedge z=x\mathbf{0}+b\mathbf{1})\big).$$

Using Example 11.1, we compute the value d of $c\mathbf{1}$, and specify z as d in the consequent. The resulting position

$$\big(|b|\leq|t|\rightarrow(|c|\leq|x|+|b|\wedge c=x+b)\big)\rightarrow\big(|b\mathbf{1}|\leq|t|\rightarrow|c\mathbf{1}|\leq|x\mathbf{0}|+|b\mathbf{1}|\wedge c\mathbf{1}=x\mathbf{0}+b\mathbf{1}\big),$$

in view of (10), is true, so we win.

The right inductive step will be handled in a similar way, only relying on (11) and (12) instead of (9) and (10). ■

Fact 12.5 **CLA4** $\vdash\sqcup z(z=x\times y)$.

Proof. The general scheme of proof here is very similar to the one employed in the proof of Fact 12.4, and details are left as an exercise to the reader. Here we shall only point out that, the four basic **PA**-provable facts that play the same role here as facts (9)-(12) in the proof of Fact 12.4 are the following:

$$\begin{aligned}
s\mathbf{0}\times r\mathbf{0}&=(s\times r)\mathbf{00}, && \text{i.e.,}\quad 2s\times 2r=4(s\times r);\\
s\mathbf{0}\times r\mathbf{1}&=(s\times r)\mathbf{00}+s\mathbf{0}, && \text{i.e.,}\quad 2s\times(2r+1)=4(s\times r)+2s;\\
s\mathbf{1}\times r\mathbf{0}&=(s\times r)\mathbf{00}+r\mathbf{0}, && \text{i.e.,}\quad (2s+1)\times 2r=4(s\times r)+2r;\\
s\mathbf{1}\times r\mathbf{1}&=(s\times r)\mathbf{00}+(s+r)\mathbf{1}, && \text{i.e.,}\quad (2s+1)\times(2r+1)=4(s\times r)+2(s+r)+1.
\end{aligned}$$

Also, where the previous proof relied on Axiom 9 and Example 11.1, the present proof, in addition, will rely on Fact 12.4. ■

The following fact establishes the provable computability of all (polynomial) functions represented through terms:

Fact 12.6 *For any term τ (not containing z),* **CLA4** $\vdash\sqcup z(z=\tau)$.

Proof. We prove this fact by (meta)induction on the complexity of τ. The base cases are those of τ being the constant 0 or a variable x. Both of the corresponding sentences $\sqcup z(z=0)$ and $\sqcap x\sqcup z(z=x)$ are provable in **CL12** and hence also in **CLA4**. Next, assume τ is θ'. By the induction hypothesis, **CLA4** proves $\sqcup z(z=\theta)$. **CLA4** also proves $\sqcup y(y=x')$ (Axiom 8). The desired $\sqcup z(z=\theta')$ is a logical consequence of these two. The remaining cases of τ being $\theta_1+\theta_2$ or $\theta_1\times\theta_2$ are handled in a similar way, relying on Facts 12.4 and 12.5, respectively. ■

The formula of the following fact, as a computational problem, is about finding the (nonnegative) difference z between any two numbers x and y and then telling whether this difference is $x-y$ or $y-x$.

Fact 12.7 **CLA4** $\vdash \sqcup z(x=y+z \sqcup y=x+z)$.

Proof. Argue in **CLA4**. By **CLA4**-Induction on x, we want to show

$$\sqcap y\Big(|y|\le|t| \to \sqcup z\big(|z|\le|x|+|y| \wedge (x=y+z \sqcup y=x+z)\big)\Big). \tag{15}$$

The *basis* $|y|\le|t| \to \sqcup z\big(|z|\le|0|+|y| \wedge (0=y+z \sqcup y=0+z)\big)$ is obviously[9] solved by the strategy that chooses the value of y for the variable z and then selects the right $\sqcup$-disjunct.

To solve the *left inductive step*

$$\begin{array}{l}\sqcap y\Big(|y|\le|t| \to \sqcup z\big(|z|\le|x|+|y| \wedge (x=y+z \sqcup y=x+z)\big)\Big) \to \\ \sqcap y\Big(|y|\le|t| \to \sqcup z\big(|z|\le|x\mathbf{0}|+|y| \wedge (x\mathbf{0}=y+z \sqcup y=x\mathbf{0}+z)\big)\Big),\end{array} \tag{16}$$

we wait till Environment specifies a constant a for y in the consequent. Then, using Fact 12.2, we compute the binary predecessor b of a, and also figure out whether $a = b\mathbf{0}$ or $(\sqcup)$ $a=b\mathbf{1}$.

Case 1: $a=b\mathbf{0}$. We specify y as b in the antecedent of (16), this way forcing Environment to choose a constant c for z there (unless $|b|\le|t|$ is false, in which case $|a|\le|t|$ is also false and we win), and also choose one of the disjuncts of $x=b+c \sqcup b=x+c$. Using Axiom 9, we calculate the value d of $c\mathbf{0}$, and specify z as d in the consequent of (16). Further, if Environment has chosen $x=b+c$ in the antecedent, we choose the left $\sqcup$-disjunct in the consequent. This means that, by now, (16) is brought down to

$$(|b|\le|t| \to |c|\le|x|+|b| \wedge x=b+c) \to (|b\mathbf{0}|\le t \to |c\mathbf{0}|\le|x\mathbf{0}|+|b\mathbf{0}| \wedge x\mathbf{0}=b\mathbf{0}+c\mathbf{0}).$$

From **PA**, the above is true, so we win. Similarly, if Environment has chosen $b=x+c$ in the antecedent of (16), then we choose the right $\sqcup$-disjunct in the consequent, and again win.

Case 2: $a=b\mathbf{1}$. Again, we specify y as b in the antecedent of (16), this way forcing Environment to choose a constant c for z there, and also to choose one of the disjuncts of $x=b+c \sqcup b=x+c$.

Subcase 2.1: $b=x+c$ is chosen. Using Example 11.1, we calculate the value d of $c\mathbf{1}$, specify z as d in the consequent of (16), and choose the right $\sqcup$-disjunct there. By now, (16) is brought down to

$$(|b|\le|t| \to |c|\le|x|+|b| \wedge b=x+c) \to (|b\mathbf{1}|\le|t| \to |c\mathbf{1}|\le|x\mathbf{0}|+|b\mathbf{1}| \wedge b\mathbf{1}=x\mathbf{0}+c\mathbf{1}).$$

According to **PA**, the above is true, so we win.

Subcase 2.2: $x=b+c$ is chosen. First, using Example 11.3, we figure our whether $c=0$ or $(\sqcup)$ $c\ne0$.

Subsubcase 2.2.1: $c=0$. Using Axiom 8, we calculate the value d of $0'$, specify z as d in the consequent of (16), and choose the right $\sqcup$-disjunct there. This means that, by now, (16) is brought down to

$$(|b|\le|t| \to |0|\le|x|+|b| \wedge x=b+0) \to (|b\mathbf{1}|\le|t| \to |0'|\le|x\mathbf{0}|+|b\mathbf{1}| \wedge b\mathbf{1}=x\mathbf{0}+0')$$

which, by **PA**, is true, so we win.

Subsubcase 2.2.2: $c\ne0$. Using Axiom 9 and Fact 12.3, we calculate d with $c\mathbf{0}=d'$, i.e. $d=c\mathbf{0}-1$, specify z as d in the consequent of (16), and choose the left $\sqcup$-disjunct there. This means that, by now, (16) is brought down to the following true (by **PA**) position, so we win:

$$(|b|\le|t| \to |c|\le|x|+|b| \wedge x=b+c) \to \big(|b\mathbf{1}|\le|t| \to |c\mathbf{0}-1|\le|x\mathbf{0}|+|b\mathbf{1}| \wedge x\mathbf{0}=b\mathbf{1}+(c\mathbf{0}-1)\big).$$

The *right inductive step* is handled in a rather similar way, and it is left as an exercise.

Thus, we have proven (15). Now, the target sentence $\sqcap x\sqcap y\sqcup z(x=y+z \sqcup y=x+z)\big)$ can be easily seen to be a logical consequence of (15) and the **PA**-provable fact $\forall y(|y|\le|y|)$. ■

Fact 12.8 **CLA4** $\vdash x=y \sqcup x\ne y$.

[9]Here and often elsewhere implicitly relying on **PA**.

Proof. Argue in **CLA4**. In order to solve $x=y \sqcup x\neq y$, using Fact 12.7, we find the difference a between x and y. Further, using Example 11.3, we figure out whether $a=0$ or $a\neq 0$. If $a=0$, we choose the left $\sqcup$-disjunct, otherwise we choose the right $\sqcup$-disjunct. ■

For natural numbers n and i — as always identified with the corresponding binary numerals — such that $i<|n|$, in our metalanguage, we let $[n]_i$ mean bit $\#i$ of n, where the count of the bits of n starts from 0 rather than 1, and proceeds from left to right. So, for instance, if $n = 100$, then 1 is its bit $\#0$, and the 0s are its bits $\#1$ and $\#2$. We treat $[n]_i$ as a pseudoterm just like $|x|$, meaning that we can feel free to write expressions such as $[x]_y=z$, understood as abbreviations, in formulas of **CLA4**.

Fact 12.9 **CLA4** $\vdash y<|x| \rightarrow \sqcup z(z=[x]_y)$.

Proof. Argue in **CLA4**. By induction on x, we want to show

$$\sqcap y\Big(|y|\leq|x| \rightarrow \big(y<|x| \rightarrow \sqcup z(|z|\leq 0' \wedge z=[x]_y)\big)\Big). \tag{17}$$

The basis $|y|\leq|0| \rightarrow \big(y<|0| \rightarrow \sqcup z(|z|\leq 0' \wedge z=[0]_y)\big)$ is obviously solved by a strategy that makes no moves.

To solve the left inductive step

$$\sqcap y\Big(|y|\leq|x| \rightarrow \big(y<|x| \rightarrow \sqcup z(|z|\leq 0' \wedge z=[x]_y)\big)\Big) \rightarrow \sqcap y\Big(|y|\leq|x\mathbf{0}| \rightarrow \big(y<|x\mathbf{0}| \rightarrow \sqcup z(|z|\leq 0' \wedge z=[x\mathbf{0}]_y)\big)\Big),$$

we wait till Environment selects a value a for y in the consequent. Using Facts 12.1 and 12.8, we figure out whether $a=|x|$ or not. If yes, we select 0 for z in the consequent. If not, we choose a for y in the antecedent and wait till Environment responds by selecting a constant b for z there, after which we choose the same constant b for z in the consequent. With a little thought, this strategy can be seen to win.

The right inductive step has a similar strategy, with the difference that, if $a=|x|$, it chooses the value of $0'$ (found using Axiom 8) for z in the consequent.

Now, the target $y<|x| \rightarrow \sqcup z(z=[x]_y)$ can be seen to be a logical consequence of (17) and the **PA**-provable fact $\forall(y<|x| \rightarrow |y|\leq|x|)$. ■

The exponentiation function 2^x increases the size of its argument exponentially and hence, in view of the soundness of **CLA4**, cannot be provably computable. According to the following fact, however, the same is not the case for a limited version of the function:

Fact 12.10 **CLA4** *proves both of the following:*

$$x\leq|z| \rightarrow \sqcup y(y=2^x); \tag{18}$$

$$\sqcup y(y=2^{|r|}). \tag{19}$$

Proof. Argue in **CLA4**. By **CLA4**-induction on x, we want to prove $x\leq|z| \rightarrow \sqcup y(|y|\leq|z|' \wedge y=2^x)$, from which (18) immediately follows by LC.

The basis $0\leq|z| \rightarrow \sqcup y(|y|\leq|z|' \wedge y=2^0)$ is obviously solved by choosing the value a of $0'$ for y. Such an a can be found using Axiom 8. The left inductive step is

$$\big(x\leq|z| \rightarrow \sqcup y(|y|\leq|z|' \wedge y=2^x)\big) \rightarrow \big(x\mathbf{0}\leq|z| \rightarrow \sqcup y(|y|\leq|z|' \wedge y=2^{x\mathbf{0}})\big).$$

To solve it, we wait till Environment chooses a value for y in the antecedent. If such a value is never chosen, Environment loses unless $x\leq|z|$ is false. But, if $x\leq|z|$ is false, then so is $x\mathbf{0}\leq|z|$, and we win. So, assume a is the constant chosen by Environment in the antecedent for y. Using Fact 12.5, we compute b with $b=a^2$, and choose b for y in the consequent. We win because the game will have evolved to $(x\leq|z| \rightarrow |a|\leq|z|' \wedge a=2^x) \rightarrow (x\mathbf{0}\leq|z| \rightarrow |b|\leq|z|' \wedge b=2^{x\mathbf{0}})$, i.e.

$$(x\leq|z| \rightarrow |a|\leq|z|' \wedge a=2^x) \rightarrow (2x\leq|z| \rightarrow |a^2|\leq|z|' \wedge a^2=2^{2x})$$

which, by **PA**, is true. The right inductive step is similar, with the difference that here we shall choose b to be $2a^2$ rather than a^2, computing which will take Axiom 9 in addition to Fact 12.5.

Thus, (18) is proven. Now we solve (19), i.e. $\sqcup y(y{=}2^{|r|})$, as follows. First, using Fact 12.1, we find the value a of $|r|$. Next, using (18) — namely, specifying its z and x as r and a, respectively — we compute the value b with $b{=}2^a$, i.e. $b{=}2^{|r|}$, and choose that b for y in $\sqcup y(y{=}2^{|r|})$. ■

We generalize the earlier notation $[x]_y$ to $[x]^z_y$, additionally to $y{<}|x|$ requiring that $y{+}z{\le}|x|$. It means "the substring of x of length z which starts at the yth bit". For instance, if $x{=}111010$, then $[x]^3_2{=}101$, $[x]^6_0{=}x$ and, for each $i \in \{0,\dots,5\}$, $[x]^0_i{=}0$. As always, we identify the bit string $[x]^z_y$ with the number it represents in the binary notation. Note that the old $[x]_y$ is the special case of $[x]^z_y$ with $z{=}1$. The following fact states the provable computability of the function $[x]^z_y$.

Fact 12.11 **CLA4** $\vdash y{<}|x| \wedge y{+}z{\le}|x| \to \sqcup t(t{=}[x]^z_y)$.

Proof. Argue in **CLA4**. First, by **CLA4**-induction on r, we want to prove

$$y{<}|x| \wedge y{+}|r|{\le}|x| \to \sqcup t(|t|{\le}|x| \wedge t{=}[x]^{|r|}_y). \tag{20}$$

The basis $y{<}|x| \wedge y{+}|0|{\le}|x| \to \sqcup t(|t|{\le}|x| \wedge t{=}[x]^{|0|}_y)$ is solved straightforwardly by choosing 0 for t. That is because 0 stands for the empty bit string, and so does $[x]^{|0|}_y$ for whatever x, y with $y{<}|x|$.

The right inductive step is

$$\big(y{<}|x| \wedge y{+}|r|{\le}|x| \to \sqcup t(|t|{\le}|x| \wedge t{=}[x]^{|r|}_y)\big) \to \big(y{<}|x| \wedge y{+}|r\mathbf{1}|{\le}|x| \to \sqcup t(|t|{\le}|x| \wedge t{=}[x]^{|r\mathbf{1}|}_y)\big). \tag{21}$$

Assume $y{<}|x| \wedge y{+}|r\mathbf{1}|{\le}|x|$. Then we also have $y{<}|x| \wedge y{+}|r|{\le}|x|$. Using the antecedental resource $\sqcup t(|t|{\le}|x| \wedge t{=}[x]^{|r|}_y)$, we find a with $|a|{\le}|x|$ such that $a{=}[x]^{|r|}_y$. Using Facts 12.1, 12.4 and 12.9, we further find b with $b{=}[x]_{y+|r|}$. Now the sought value of t is the value of $a\mathbf{0}{+}b$, which we compute using Axiom 9 and Fact 12.4.

To solve the left inductive step, first we figure out (using Example 11.3) whether $r{=}0$ or not. If $r{=}0$, we ignore the antecedent and act in the consequent as the strategy for the basis of induction did. Otherwise, we act as the strategy for the right inductive step did.

(20) is thus proven. To solve the target $y{<}|x| \wedge y{+}z{\le}|x| \to \sqcup t(t{=}[x]^z_y)$, assume $y{<}|x| \wedge y{+}z{\le}|x|$. Then $z{\le}|x|$. Using Facts 12.2 and 12.10, we find the value a of the binary predecessor of 2^z. Note that $|a|{=}z$. Now, using (20), we find the value b of $[x]^{|a|}_y$, i.e. of $[x]^z_y$. Selecting b for t solves the problem. ■

13 The soundness of CLA4

This section is devoted to proving the soundness part of Theorem 11.5. It means showing that any **CLA4**-provable sentence X (as always, identified with its standard interpretation $X^\dagger$) has a polynomial time solution, and that, furthermore, such a solution for X, together with an explicit polynomial bound τ for its time complexity, can be efficiently extracted from any extended **CLA4**-proof of X. Consider any sentence X with a fixed **CLA4**-proof.

For presentational considerations, by induction on the length of the proof of X, we will first simply show that a polynomial time solution of X exists. Only after that, at the end of this section, we will show that such a solution, together with an explicit polynomial bound for its time complexity, is or can be constructed efficiently.

Assume X is an axiom of **CLA4**. If X is a Peano axiom, then it is a true elementary sentence and therefore is won by a machine that makes no moves. If X is $\sqcap x \sqcup y(y{=}x')$ (Axiom 8), then it is won by a machine that (for the constant x chosen by Environment for the variable x) computes the value a of $x{+}1$, and makes a as its only move in the play. Similarly, if X is $\sqcap x \sqcup y(y{=}x\mathbf{0})$ (Axiom 9), it is won by a machine that computes the value a of $2x$, and makes a as its only move in the play. Needless to point out that all of the above machines run in polynomial time.

Next, suppose X is obtained from premises $Y_1,\dots,Y_n$ by LC. By the induction hypothesis, for each $i \in \{1,\dots,n\}$, we already have a solution (HPM) $\mathcal{N}_i$ of Y_i together with an explicit polynomial bound ξ_i for the time complexity of $\mathcal{N}_i$. Of course, every such HPM $\mathcal{N}_i$ can as well be seen as an n-ary GHPM that simply ignores its inputs. Then, by Theorem 10.1, we can (efficiently) construct an n-ary GHPM $\mathcal{M}$, together with

an explicit polynomial bound $\tau(\xi_1,\ldots,\xi_n)$ for the time complexity of the HPM $\mathcal{M}(\ulcorner\mathcal{N}_1\urcorner,\ldots,\ulcorner\mathcal{N}_n\urcorner)$ such that the latter solves X.

Finally, suppose X is (the $\sqcap$-closure of) $F(x)$, where $F(x)$ is a polynomially bounded formula, and X is obtained by **CLA4**-Induction on x. So, the premises are (the $\sqcap$-closures of) $F(0)$, $F(x)\to F(x\mathbf{0})$ and $F(x)\to F(x\mathbf{1})$. By the induction hypothesis, there are HPMs $\mathcal{N}$, $\mathcal{K}_0$,$\mathcal{K}_1$ — with explicit polynomial bounds ξ,ζ_0,ζ_1 for their time complexities, respectively — that solve these three premises, respectively. Fix them.

We need certain auxiliary concepts. Consider any polynomially bounded formula H, any legal position Φ of $\sqcap H$, and any legal move α (by whichever player $\wp$) in position Φ. In this context, we say that α is **unreasonable** if it signifies a choice of a constant c for a variable y in a $\sqcup y\big(S(y,\vec{z})\wedge G\big)$ or $\sqcap y\big(S(y,\vec{z})\to G\big)$ (depending on whether $\wp=\top$ or $\wp=\bot$) subcomponent of H, such that c violates the conditions on its size imposed by the sizebound $S(y,z)$. Rather than trying to turn this otherwise clear intuitive explanation into a strict definition, providing an example would be sufficient. Let H be the formula $0{=}0\wedge\sqcup y(|y|\leq|z|'\wedge z{=}y\mathbf{0})$. Then the move 1.1111 is unreasonable in position $\langle\bot 11\rangle$. That is because this move signifies choosing the constant 1111 for y. And the move $\bot 11$ of the position has set the value of z to 11 and hence the value of $|z|$ to 2. So, the condition $|y|\leq|z|'$, i.e. $|1111|\leq|11|'$, is violated. Any other move $1.n$ with $|n|>3$, such as 1.1000 or 1.111111111, would also be unreasonable in that position.

We replace $\mathcal{N}$ by its "**reasonable counterpart**" $\mathcal{N}'$ — an HPM which never makes unreasonable moves but otherwise is essentially the same as $\mathcal{N}$. Namely, $\mathcal{N}'$ is a machine that works just like $\mathcal{N}$, with the difference that, every time $\mathcal{N}$ makes an unreasonable move that chooses some (offensively long) constant c for a variable bound by a (bounded) quantifier of $F(0)$, $\mathcal{N}'$ chooses (the always safe) 0 instead. Note that this does not decrease the chances of the machine to win, as unreasonable moves always result in the corresponding subgames' being lost, anyway. Obviously $\mathcal{N}'$ can be efficiently constructed from $\mathcal{N}$. Further, the corresponding explicit polynomial bound ξ' can also be efficiently indicated (the latter will depend on ξ and the sizebounds of the $\sqcup$-bound variables of $F(0)$). In a similar fashion, we replace $\mathcal{K}_0$,$\mathcal{K}_1$ by their "reasonable counterparts" $\mathcal{K}'_0$,$\mathcal{K}'_1$ and the corresponding explicit polynomial bounds ζ'_0,ζ'_1 for their time complexities. For simplicity, we further replace the three bounds ξ',ζ'_0,ζ'_1 by the (generously taken) common bound $\phi{=}\xi'{+}\zeta'_0{+}\zeta'_1$ for the time complexities of all three machines $\mathcal{N}'$, $\mathcal{K}'_0$ and $\mathcal{K}'_1$.

We now describe an HPM $\mathcal{M}$ that solves the conclusion $F(x)$. In this description, we use the term "**synchronizing**" to mean applying copycat between two (sub)games of the form A and $\neg A$. This means copying one player's moves in A as the other player's moves in $\neg A$, and vice versa. The effect achieved this way is that the games to which A and $\neg A$ eventually evolve (the final positions hit by them, that is) will be of the form A' and $\neg A'$, that is, one will remain the negation of the other, so that one will be won by a given player iff the other is lost by the same player. **Moderated synchronization** means the same, with the only difference that, whenever a player makes an unreasonable move by choosing an (offensively long) constant c for a variable bound by a bounded quantifier, the move is copied by the synchronizer with c replaced by 0.

Throughout our description and analysis of the work of $\mathcal{M}$, we assume that its adversary never makes illegal moves, for otherwise $\mathcal{M}$ easily detects illegal behavior and retires with victory.

At the beginning, $\mathcal{M}$ waits for Environment to choose constants for the free variables of $F(x)$. Assume k is the length of the constant chosen for the variable x, and the bits of that constant, in the left-to-right order, are $b_1,b_2,\ldots b_k$. We shall also assume here that $k{\neq}0$, for otherwise the case is straightforward. Let d_0 be the constant 0 and, for each $i\in\{1,\ldots,k\}$, let d_i be the constant $b_1\ldots b_i$. So, the constant chosen by Environment for x is d_k. For each $i\in\{1,\ldots,k\}$, let $\mathcal{K}'_{b_i}$ stand for $\mathcal{K}'_0$ if $b_i=0$, and for $\mathcal{K}'_1$ if $b_i=1$. Similarly, let xb_i stand for $x\mathbf{0}$ if $b_i=0$, and for $x\mathbf{1}$ if $b_i=1$.

After Environment chooses constants for all free variables of $F(x)$, the work of $\mathcal{M}$ consists in continuously polling its run tape to see if Environment has made any new moves, combined with simulating, in parallel, one play of $\sqcap\big(F(0)\big)$ by $\mathcal{N}'$ and — for each $i\in\{1,\ldots,k\}$ — one play of $\sqcap\big(F(x)\to F(xb_i)\big)$ by $\mathcal{K}'_{b_i}$. In the simulation of $\mathcal{N}'$, $\mathcal{M}$ lets the imaginary adversary of $\mathcal{N}'$ choose, at the very beginning of the play, the same constants for the free variables of $F(0)$ as $\mathcal{M}$'s adversary chose for those variables in the real play. In the simulation of each $\mathcal{K}'_{b_i}$, $\mathcal{M}$ lets the imaginary adversary of $\mathcal{K}'_{b_i}$ choose, at the very beginning of the play, the constant d_{i-1} for x and the same constants for all other free variables of $F(x)\to F(xb_i)$ as $\mathcal{M}$'s adversary chose for those variables in the real play.

Let $F'(x)$ be the result of substituting (see Definition 4.2) in $F(x)$ each free variable of $F(x)$ other than x by the constant chosen by (the real) Environment for that variable. Thus, after Environment's initial moves, $\sqcap\big(F(x)\big)$ has been brought down to $F'(d_k)$. Similarly, after the initial moves by the imaginary adversary of

$\mathcal{N}'$, $\sqcap\big(F(0)\big)$ will be brought down to $F'(0)$. And similarly, for each $i \in \{1,\ldots,k\}$, after the initial moves by the imaginary adversary of $\mathcal{K}'_{b_i}$, $\sqcap\big(F(x) \to F(xb_i)\big)$ will be brought down to $F'(d_{i-1}) \to F'(d_i)$.

What $\mathcal{M}$ does after the above initial moves in the real and simulated plays is that it synchronizes $k+1$ pairs of (sub)games, real or imaginary. Namely:

- It synchronizes — in the *moderated* sense — the consequent of the imaginary play of $F'(d_{k-1}) \to F'(d_k)$ by $\mathcal{K}'_{b_k}$ with the real play of $F'(d_k)$.
- For each $i \in \{1,\ldots,k-1\}$, it synchronizes the consequent of the imaginary play of $F'(d_{i-1}) \to F'(d_i)$ by $\mathcal{K}'_{b_i}$ with the antecedent of the imaginary play of $F'(d_i) \to F'(d_{i+1})$ by $\mathcal{K}'_{b_{i+1}}$.
- It synchronizes the imaginary play of $F'(0)$ (i.e. of $F'(d_0)$) by $\mathcal{N}'$ with the antecedent of the imaginary play of $F'(d_0) \to F'(d_1)$ by $\mathcal{K}'_{b_1}$.

Below is an illustration of such synchronization arrangements — indicated by arcs — for the case $d_k = d_4 = 1001$:

machine:	*status:*	*game:*
$\mathcal{N}'$	imaginary	$F'(0)$
$\mathcal{K}'_1$	imaginary	$F'(0) \to F'(1)$
$\mathcal{K}'_0$	imaginary	$F'(1) \to F'(10)$
$\mathcal{K}'_0$	imaginary	$F'(10) \to F'(100)$
$\mathcal{K}'_1$	imaginary	$F'(100) \to F'(1001)$
$\mathcal{M}$	real	$F'(1001)$

This completes our description of $\mathcal{M}$. Remembering our assumption that ($\mathcal{N}, \mathcal{K}_0, \mathcal{K}_1$ and hence) $\mathcal{N}', \mathcal{K}'_0, \mathcal{K}'_1$ win the corresponding games, with a little thought it can be seen that $\mathcal{M}$ wins $F'(d_k)$ and hence $\sqcap\big(F(x)\big)$, as desired. It now remains to show that the time complexity of $\mathcal{M}$ is also as desired.

Remembering that the machines $\mathcal{N}', \mathcal{K}'_0, \mathcal{K}'_1$ are "reasonable" and that the synchronization between the real play of $F'(d_k)$ and the consequent of $F'(d_{k-1}) \to F'(d_k)$ is moderated, one can easily write a term $\eta(\ell)$ with a single variable ℓ such that, if ℓ is greater than or equal to the size of any of the constants chosen by Environment for the free variables of $F(x)$, the sizes of no moves ever made by $\mathcal{M}$ or the simulated $\mathcal{N}', \mathcal{K}'_0, \mathcal{K}'_1$ exceed $\eta(\ell)$. For instance, if $F(x)$ is $\sqcup u\big(|u| \leq |x| \times |z| \wedge \sqcap v(|v| \leq |u| + |x| \to G)\big)$ where G is elementary, then $\eta(\ell)$ can be taken to be $\ell \times \ell + \ell + 0''''$ (here $0''''$ is to account for the size of the prefix ".", "1." or "0.1." that any legal move by $\top$ in any of the plays that we consider would take).

For the rest of this proof, pick and fix an arbitrary play (computation branch) of $\mathcal{M}$, and an arbitrary clock cycle $\mathfrak{c}$ on which $\mathcal{M}$ makes a move α in the real play of $F(x)$. Let $\hbar$ and ℓ be the timecost and the background (see Section 7) of this move, respectively. Let $d_0, \ldots, d_k$ be as in the description of the work of $\mathcal{M}$. Note that ℓ is not smaller than the size of the greatest of the constants chosen by Environment for the free variables of $F(x)$. Hence, where η is as in the previous paragraph, we have:

The sizes of no moves ever made by $\mathcal{M}$ or the simulated $\mathcal{N}', \mathcal{K}'_0, \mathcal{K}'_1$ exceed $\eta(\ell)$. (22)

The polling, simulation and copycat performed by $\mathcal{M}$ do impose some time overhead. But the latter is only (fixed) polynomial and, in our subsequent analysis, can be safely ignored. Namely, for the sake of simplicity, we are going to pretend that $\mathcal{M}$ copies moves in its copycat routine instantaneously (as soon as detected), and that the times that $\mathcal{M}$ ever spends "thinking" about what move to make are the times during which it is waiting for simulated machines to make one or several moves. Furthermore, we will pretend that the polling and the several simulations happen in a truly parallel fashion, in the sense that $\mathcal{M}$ spends a

single clock cycle on tracing a single computation step of *all* $k+1$ machines simultaneously, as well as on checking out its run tape to see if Environment has made a new move.

Let $\beta_1, \ldots, \beta_m$ be the moves by simulated machines that $\mathcal{M}$ detects by time $\mathfrak{c}$, arranged according to the times $t_1 \leq \ldots \leq t_m$ of their detections (which, by our simplifying assumptions, coincide with the timestamps of those moves in the corresponding simulated plays). Let $d = \mathfrak{c} - \hbar$. Let j be the smallest integer among $1, \ldots, m$ such that $t_j \geq d$. Since each simulated machine runs in time ϕ, in view of (22) it is clear that $t_j - d$ does not exceed $\phi(\eta(\ell))$. Nor does $t_i - t_{i-1}$ for any i with $j < i \leq m$. Therefore $t_m - d \leq (m - j + 1) \times \phi(\eta(\ell))$. Since $m, j \geq 1$, let us be generous and simply say that $t_m - d \leq m \times \phi(\eta(\ell))$. But notice that β_m is a move made by $\mathcal{K}'_{b_k}$ in the consequent of $F'(d_{k-1}) \to F(d_k)$, immediately (by our simplifying assumptions) copied by $\mathcal{M}$ in the real play when it made its move α. In other words, $\mathfrak{c} = t_m$. And $\mathfrak{c} - d = \hbar$. So, $\hbar$ does not exceed $m \times \phi(\eta(\ell))$. And, by (22), the size of α does not exceed $m \times \phi(\eta(\ell))$, either. But observe that $k \leq \ell$, and that m cannot exceed $k+1$ times the depth (see Section 4) $\mathfrak{d}$ of $F(0)$; therefore, $m \leq \mathfrak{d} \times (\ell + 1)$. Thus, (as long as we pretend that there is no polling/simulation/copycat overhead) neither the timecost nor the size of α exceed $\mathfrak{d} \times (\ell + 1) \times \phi(\eta(\ell))$.

An upper bound for the above function $\mathfrak{d} \times (\ell + 1) \times \phi(\eta(\ell))$, even after "correcting" the latter so as to precisely account for the so far suppressed polling/simulation/copycat overhead, can be written as an explicit polynomial function τ. The latter expresses the sought polynomial bound for the time complexity of $\mathcal{M}$.

Thus, we have shown how to construct, from a proof of X, an HPM $\mathcal{M}$ and an explicit polynomial function τ such that $\mathcal{M}$ solves X in time τ. Obviously our construction is effective. It remains to see that it also is — or, at least, can be made — efficient. Of course, at every step of our inductive construction (for each sentence of the proof, that is), the solution $\mathcal{M}$ of the step and its time complexity bound τ is obtained efficiently from previously constructed $\mathcal{M}$s and τs. This, however, does not guarantee that the entire construction will be efficient as well. For instance, if the proof has n steps and the size of each HPM $\mathcal{M}$ that we construct for each step is twice the size of the previously constructed HPMs, then the size of the eventual HPM will exceed 2^n and thus the construction will not be efficient, even if each of the n steps of it is so.

A trick that we can use to avoid an exponential growth of the sizes of the machines that we construct and thus achieve the efficiency of the entire construction is to deal with GHPMs instead of HPMs. Namely, assume the proof of X is the sequence $X_1, \ldots, X_n$ of sentences, with $X = X_n$. Let $\mathcal{M}_1, \ldots, \mathcal{M}_n$ be the HPMs constructed as we constructed $\mathcal{M}$s earlier at the corresponding steps of our induction. Remember that each such $\mathcal{M}_i$ was defined in terms of $\mathcal{M}_{j_1}, \ldots, \mathcal{M}_{j_k}$ for some $j_1, \ldots, j_k < i$. For simplicity and uniformity, we may just as well say that each $\mathcal{M}_i$ was defined in terms of all $\mathcal{M}_1, \ldots, \mathcal{M}_n$, with those $\mathcal{M}_j$s that were not among $\mathcal{M}_{j_1}, \ldots, \mathcal{M}_{j_k}$ simply ignored in the description of the work of $\mathcal{M}_i$. Now, for each such $\mathcal{M}_i$, let $\mathcal{M}'_i$ be the n-ary GHPM whose description is obtained from that of $\mathcal{M}_i$ by replacing each reference to (any previously constructed) $\mathcal{M}_j$ by "$\mathcal{M}'_j(\ulcorner\mathcal{M}'_1\urcorner, \ldots, \ulcorner\mathcal{M}'_n\urcorner)$ where, for each $e \in \{1, \ldots, n\}$, $\mathcal{M}'_e$ is the machine encoded by the eth input".[10] As it is easy to see by induction on i, $\mathcal{M}_i$ and $\mathcal{M}'_i(\ulcorner\mathcal{M}'_1\urcorner, \ldots, \ulcorner\mathcal{M}'_n\urcorner)$ are essentially the same, in the sense that our earlier analysis of the play and time complexity of the former applies to the latter just as well. So, $\mathcal{M}'_n(\ulcorner\mathcal{M}'_1\urcorner, \ldots, \ulcorner\mathcal{M}'_n\urcorner)$ wins X_n, i.e. X. At the same time, note that the size of each GHPM $\mathcal{M}'_i$ is independent of the sizes of the other (previously constructed) GHPMs. Based on this fact, with some analysis, one can see that then the HPM $\mathcal{M}'_n(\ulcorner\mathcal{M}'_1\urcorner, \ldots, \ulcorner\mathcal{M}'_n\urcorner)$ is indeed constructed efficiently.

As for the explicit polynomial bounds $\tau_1, \ldots, \tau_n$ for the time complexities of the n HPMs $\mathcal{M}'_1(\ulcorner\mathcal{M}'_1\urcorner, \ldots, \ulcorner\mathcal{M}'_n\urcorner)$, $\ldots$, $\mathcal{M}'_n(\ulcorner\mathcal{M}'_1\urcorner, \ldots, \ulcorner\mathcal{M}'_n\urcorner)$, their sizes can be easily seen to be polynomial in the size of the proof. That is because, for each $i \in \{1, \ldots, n\}$, the size of τ_i only increases the sizes of the earlier constructed τ_js by adding (rather than multiplying by) a certain polynomial quantity.[11] Thus, the explicit bound τ_n for the time complexity of the eventual HPM $\mathcal{M}'_n(\ulcorner\mathcal{M}_1\urcorner, \ldots, \ulcorner\mathcal{M}_n\urcorner)$ is indeed constructed efficiently.

14 The extensional completeness of CLA4

This section is devoted to proving the completeness part of Theorem 11.5. It means showing that, for any arithmetical problem A that has a polynomial time solution, there is a theorem of **CLA4** which, under the

[10] For simplicity, here we assume that every number is a code of some n-ary GHPM; alternatively, $\mathcal{M}'_i$ can be defined so that it does nothing if any of its relevant inputs is not the code of some n-ary GHPM.

[11] The fact that we represent complexity bounds as explicit polynomial functions rather than polynomial tree-terms or even graph-terms is relevant here.

standard interpretation, equals ("expresses") A.

14.1 X, $\mathcal{X}$ and χ

So, let us pick an arbitrary polynomial-time-solvable arithmetical problem A. By definition, A is an arithmetical problem because, for some sentence X of the language of **CLA4**, $A = X^\dagger$. For the rest of this section, we fix such a sentence X, and fix $\mathcal{X}$ as an HPM that solves A (and hence $X^\dagger$) in polynomial time. Specifically, we assume that $\mathcal{X}$ runs in time χ, where χ, which we also fix for the rest of this section, is a single-variable term of the language of **PA** — and hence can as well be seen/written as an explicit polynomial function — with $\chi(x) \geq x$ for all x. For readability, we also agree that, throughout the rest of this section, "**formula**" exclusively means a subformula of X, in which some variables may be renamed.

X may not necessarily be provable in **CLA4**, and our goal is to construct another sentence $\overline{X}$ for which, just like for X, we have $A = \overline{X}^\dagger$ and which, perhaps unlike X, is provable in **CLA4**.

Remember our convention about identifying formulas of the language of **CLA4** with (the games that are) their standard interpretations. So, in the sequel, just as we have done so far, we shall typically write $E, F, \ldots$ to mean either $E, F, \ldots$ or $E^\dagger, F^\dagger, \ldots$. Similar conventions apply to terms as well. In fact, we have already used this convention when saying that $\mathcal{X}$ runs in time χ. What was really meant was that it runs in time $\chi^\dagger$.

14.2 Preliminary insights

Our proof is a little long and, in the process of going through it, it is easy to get lost in the forest and stop seeing it for the trees. Therefore, it might be worthwhile to try to get some preliminary insights into the basic idea behind this proof before venturing into its details.

Let us consider the simplest nontrivial special case where X is

$$\sqcap x\big(Y(x) \sqcup Z(x)\big)$$

for some elementary formulas $Y(x)$ and $Z(x)$ (perhaps $Z(x)$ is $\neg Y(x)$, in which case X expresses an ordinary decision problem — the problem of deciding the predicate $Y(x)$).

The assertion "$\mathcal{X}$ does not win X in time χ" can be formalized in the language of **PA** through as a certain sentence $\mathbb{L}$. Then we let the earlier mentioned $\overline{X}$ be the sentence

$$\sqcap x\Big(\big(Y(x) \vee \mathbb{L}\big) \sqcup \big(Z(x) \vee \mathbb{L}\big)\Big).$$

Since $\mathcal{X}$ *does* win the game X in time χ, $\mathbb{L}$ is false. Hence $Y(x) \vee \mathbb{L}$ is equivalent to $Y(x)$, and $Z(x) \vee \mathbb{L}$ is equivalent to $Z(x)$. This means that $\overline{X}$ and X, as games, are the same, that is, $\overline{X}^\dagger = X^\dagger$. It now remains to understand why **CLA4** $\vdash \overline{X}$.

A central lemma here is one establishing that the work of $\mathcal{X}$ is "*provably traceable*". Roughly, in our present case this means the provability of the fact that, for any ($\sqcap$) value chosen for x by Environment — let us continue referring to that value as x — we can tell ($\sqcup$) the configuration of $\mathcal{X}$ in the corresponding play of $\overline{X}$ at any given time t. Letting $\mathcal{X}$ work for $\chi(x)$ steps, one of the following four eventual scenarios should take place, and the provable traceability of the work of $\mathcal{X}$ can be shown to imply that **CLA4** proves the $\sqcup$-disjunction of sentences describing those scenarios:

Scenario 1: $\mathcal{X}$ makes the move 0 (and no other moves).

Scenario 2: $\mathcal{X}$ makes the move 1 (and no other moves).

Scenario 3: $\mathcal{X}$ does not make any moves.

Scenario 4: $\mathcal{X}$ makes an illegal move (perhaps after first making a legal move 0 or 1).

In the case of Scenario 1, the play over $\overline{X}$ hits $Y(x) \vee \mathbb{L}$. And **CLA4** — in fact, **PA** — proves that, in this case, $Y(x) \vee \mathbb{L}$ is true. The truth of $Y(x) \vee \mathbb{L}$ is indeed very easily established: if it was false, then $Y(x)$ should be false, but then the play of $\mathcal{X}$ over X hits the false $Y(x)$ and hence is lost, but then $\mathbb{L}$ is true, but then

$Y(x) \vee \mathbb{L}$ is true. Thus, **CLA4** $\vdash$ (*Scenario 1*) $\to Y(x) \vee \mathbb{L}$, from which, by LC, **CLA4** $\vdash$ (*Scenario 1*) $\to \overline{X}$. The case of Scenario 2 is symmetric.

In the case of Scenario 3, (**CLA4** proves that) $\mathcal{X}$ loses, i.e. $\mathbb{L}$ is true, and hence, say, $Y(x) \vee \mathbb{L}$ (or $Z(x) \vee \mathbb{L}$ if you like) is true. That is, **CLA4** $\vdash$ (*Scenario 3*) $\to Y(x) \vee \mathbb{L}$, from which, by LC, **CLA4** $\vdash$ (*Scenario 3*) $\to \overline{X}$. The case of Scenario 4 is similar.

Thus, for each $i \in \{1, 2, 3, 4\}$, **CLA4** $\vdash$ (*Scenario i*) $\to \overline{X}$. And, as mentioned, we also have

$$\textbf{CLA4} \vdash (\textit{Scenario 1}) \sqcup (\textit{Scenario 2}) \sqcup (\textit{Scenario 3}) \sqcup (\textit{Scenario 4}).$$

The desired **CLA4** $\vdash \overline{X}$ follows from the above provabilities by LC.

The above was about the pathologically simple case of $X = \sqcap x\big(Y(x) \sqcup Z(x)\big)$, and the general case will be much more complex, of course. Among other things, showing the provability of $\overline{X}$ would require a certain metainduction on its complexity, which we did not need in the present case. But the idea that we have just tried to explain would still remain valid and central, only requiring certain — nontrivial yet doable — adjustments and refinements.

14.3 The sentence $\mathbb{L}$

By a **literal** we mean $\top$, $\bot$, or an atomic formula with or without negation $\neg$. By a **politeral** of a formula we mean a positive (not in the scope of $\neg$) occurrence of a literal in it. For instance, the occurrence of p, as well as of $\neg q$ (but not q), is a politeral of $p \wedge \neg q$. While a politeral is not merely a literal but a literal L *together* with a fixed occurrence, we shall often refer to it just by the name L of the literal, assuming that it is clear from the context which (positive) occurrence of L is meant.

We assume that the reader is sufficiently familiar with Gödel's technique of encoding and arithmetizing. Using that technique, we can construct a sentence $\mathbb{L}$ of the language of **PA** which asserts "$\mathcal{X}$ does not win X in time χ". Namely, let $E_1(\vec{x}), \ldots, E_n(\vec{x})$ be all subformulas of X, where all free variables of each $E_i(\vec{x})$ are among $\vec{x}$ (but not necessarily vice versa). Then $\mathbb{L}$ is the $\vee$-disjunction of natural formalizations of the following statements:

1. *There is a $\top$-illegal position of X spelled on the run tape of $\mathcal{X}$ on some clock cycle of some computation branch of $\mathcal{X}$.*
2. *There is a clock cycle c in some computation branch of $\mathcal{X}$ on which $\mathcal{X}$ makes a move whose timecost exceeds $\chi(\ell)$, where ℓ is the background of c.*
3. *There is a (finite) legal run Γ of X generated by $\mathcal{X}$ and a tuple $\vec{c}$ of constants ($\vec{c}$ of the same length as $\vec{x}$) such that:*
   - $\langle\Gamma\rangle X = E_1(\vec{c})$*, and we have* $\neg\|E_1(\vec{c})\|$ *(i.e.,* $\|E_1(\vec{c})\|$ *is false),*
   - *or ..., or*
   - $\langle\Gamma\rangle X = E_n(\vec{c})$*, and we have* $\neg\|E_n(\vec{c})\|$ *(i.e.,* $\|E_n(\vec{c})\|$ *is false).*

14.4 The overline notation

As we remember, our goal is to construct a formula $\overline{X}$ which expresses the same problem as X does and which is provable in **CLA4**. For any formula E — including X — we let $\overline{E}$ be the result of replacing in E every politeral L by $L \vee \mathbb{L}$.

Lemma 14.1 *Any literal L is equivalent (in the standard model of arithmetic) to $L \vee \mathbb{L}$.*

Proof. That L implies $L \vee \mathbb{L}$ is immediate, as the former is a disjunct of the latter. For the opposite direction, suppose $L \vee \mathbb{L}$ is true at a given valuation e. Its second disjunct cannot be true, because $\mathcal{X}$ *does* win X in time χ, contrary to what $\mathbb{L}$ asserts. So, the first disjunct, i.e. L, is true. ■

Lemma 14.2 *For any formula E, including X, we have $E^\dagger = \overline{E}^\dagger$.*

Proof. Immediately from Lemma 14.1 by induction on the complexity of E. ■

In view of the above lemma, what now remains to do for the completion of our completeness proof is to show that $\mathbf{CLA4} \vdash \overline{X}$. The rest of the present section is entirely devoted to this task.

Lemma 14.3 *For any formula E,* $\mathbf{CLA4} \vdash \mathbb{L} \to \forall\overline{E}$.

Proof. Induction on the complexity of E. The base, which is about the cases where E is a literal, is straightforward, as then $\mathbb{L}$ is a disjunct of $\overline{E}$. If E has the form $H_0 \wedge H_1$, $H_0 \vee H_1$, $H_0 \sqcap H_1$ or $H_0 \sqcup H_1$ then, by the induction hypothesis, **CLA4** proves $\mathbb{L} \to \forall\overline{H_0}$ and $\mathbb{L} \to \forall\overline{H_1}$, from which $\mathbb{L} \to \forall\overline{E}$ follows by LC. Similarly, if E has the form $\forall xH(x)$, $\exists xH(x)$, $\sqcap xH(x)$ or $\sqcup xH(x)$, then, by the induction hypothesis, **CLA4** proves $\mathbb{L} \to \forall\overline{H(x)}$, from which $\mathbb{L} \to \forall\overline{E}$ follows by LC. ■

14.5 The single-circle and double-circle notations

The way we encode configurations through natural numbers will be precisely described later in Section A.1. For now, it would be sufficient to say that the size of the code of a configuration is always greater than the background (see Section 7) of the corresponding clock cycle in the corresponding play. For readability, we will often identify configurations with their codes and say something like "a is a configuration" when what is precisely meant is "a is the code of a configuration".

By a **legitimate configuration** we shall mean a configuration of $\mathcal{X}$ that might have occurred in some computation branch B of $\mathcal{X}$ such that the run spelled by B is a legal run of X. The **yield** of such a configuration is the game $\langle\Phi\rangle X$, where Φ is the position spelled on the run tape in that configuration.

By a **deterministic successor** of a legitimate configuration x we mean the configuration y such that y immediately follows x (in one transition) in the scenario where Environment does not move during the cycle described by x. For $n \geq 0$, the n**th deterministic successor** of x is defined inductively by stipulating that the 0th deterministic successor of x is x, and the $(n+1)$th deterministic successor of x is the deterministic successor of the nth deterministic successor of x.

Let $E(\vec{s})$ be a formula all of whose free variables are among $\vec{s}$ (but not necessarily vice versa), and z be a variable not among $\vec{s}$. We will write $E^\circ(z, \vec{s})$ to denote an elementary formula whose free variables are z and those of $E(\vec{s})$, and which is a natural arithmetization of the predicate that, for any constants $a, \vec{c}$ in the roles of $z, \vec{s}$, holds (that is, $E^\circ(a, \vec{c})$ is true) iff a is a legitimate configuration and its yield is $E(\vec{c})$. Further, we will write $E^\circ_\circ(z, \vec{s})$ to denote an elementary formula whose free variables are z and those of $E(\vec{s})$, and which is a natural arithmetization of the predicate that, for any constants $a, \vec{c}$ in the roles of $z, \vec{s}$, holds iff $E^\circ(a, \vec{c}) \wedge E^\circ(b, \vec{c})$ is true, where b is the $\chi(|a|)$th deterministic successor of a.

We say that a formula E is **critical** iff one of the following conditions is satisfied:

- E is of the form $G_0 \sqcup G_1$ or $\sqcup yG$;
- E is of the form $\forall yG$ or $\exists yG$, and G is critical;
- E is of the form $G_0 \vee G_1$, and both G_0 and G_1 are critical;
- E is of the form $G_0 \wedge G_1$, and at least one of G_0, G_1 is critical.

Lemma 14.4 *Assume $E(\vec{s})$ is a non-critical formula all of whose free variables are among $\vec{s}$. Then*

$$\mathbf{PA} \vdash \forall\big(E^\circ_\circ(z, \vec{s}) \to \|\overline{E(\vec{s})}\|\big).$$

Proof. Assume the conditions of the lemma. Argue in **PA**. Consider arbitrary ($\forall$) values of z and $\vec{s}$, which we continue writing as z and $\vec{s}$. Suppose, for a contradiction, that $E^\circ_\circ(z, \vec{s})$ is true but $\|\overline{E(\vec{s})}\|$ is false. The falsity of $\|\overline{E(\vec{s})}\|$ implies the falsity of $\|E(\vec{s})\|$. This is so because the only difference between the two formulas is that, wherever the latter has some politeral L, the former has a $\vee$-disjunction containing L as a disjunct.

The truth of $E^\circ_\circ(z, \vec{s})$ implies that $\mathcal{X}$ reaches the configuration (computation step) z and, in the scenario where Environment does not move, $\mathcal{X}$ does not move either for at least $\chi(|z|)$ steps afterwards. If $\mathcal{X}$ does not move even after $\chi(|z|)$ steps, then it has lost the game, because the eventual position hit by the latter is $E(\vec{s})$ and the elementarization of the latter is false (it is not hard to see that every such game is indeed

lost). And if $\mathcal{X}$ does make a move sometime after $\chi(|z|)$ steps, then it violates its time complexity bound χ, because the background of that move is smaller than $|z|$ but the timecost is at least $\chi(|z|)$. Thus, in either case, $\mathcal{X}$ does not win X in time χ, that is,

$$\mathbb{L} \textit{ is true.} \tag{23}$$

Consider any non-critical formula G. By induction on the complexity of G, we are going to show that $\|\overline{G}\|$ is true for any ($\forall$) values of its free variables. Indeed:

If G is a literal, then $\|\overline{G}\|$ is $G \vee \mathbb{L}$ which, by (23), is true.

If G is $H_0 \sqcap H_1$ or $\sqcap x H(x)$, then $\|\overline{G}\|$ is $\top$ and is thus true.

G cannot be $H_0 \sqcup H_1$ or $\sqcup x H(x)$, because then it would be critical.

If G is $\forall y H(y)$ or $\exists y H(y)$, then $\|\overline{G}\|$ is $\forall y \|\overline{H(y)}\|$ or $\exists y \|\overline{H(y)}\|$. In either case $\|\overline{G}\|$ is true because, by the induction hypothesis, $\|\overline{H(y)}\|$ is true for every value of its free variables, including variable y.

If G is $H_0 \wedge H_1$, then both H_0 and H_1 are non-critical. Hence, by the induction hypothesis, both $\|\overline{H_0}\|$ and $\|\overline{H_1}\|$ are true. Hence so is $\|\overline{H_0}\| \wedge \|\overline{H_1}\|$ which, in turn, is nothing but $\|\overline{G}\|$.

Finally, if G is $H_0 \vee H_1$, then one of the formulas H_i is non-critical. Hence, by the induction hypothesis, $\|\overline{H_i}\|$ is true. Hence so is $\|\overline{H_0}\| \vee \|\overline{H_1}\|$ which, in turn, is nothing but $\|\overline{G}\|$.

Thus, for any non-critical formula G, $\|\overline{G}\|$ is true. This includes the case $G = E(\vec{s})$ which, however, contradicts our assumption that $\|E(\vec{s})\|$ is false. ■

Lemma 14.5 *Assume $E(\vec{s})$ is a critical formula all of whose free variables are among $\vec{s}$. Then*

$$\mathbf{CLA4} \vdash \exists E^\circ_\circ(z, \vec{s}) \to \forall \overline{E(\vec{s})}. \tag{24}$$

Proof. Assume the conditions of the lemma. By induction on complexity, one can easily see that the elementarization of any critical formula is false. Thus, for whatever ($\forall$) values of $\vec{s}$, $\|E(\vec{s})\|$ is false. Arguing further as we did in the proof of Lemma 14.4 when deriving (23), we find that, if $E^\circ_\circ(z, \vec{s})$ is true for whatever ($\exists$) values of z and $\vec{s}$, then so is $\mathbb{L}$. And this argument can be formalized in **PA**, so that we have $\mathbf{PA} \vdash \exists E^\circ_\circ(z, \vec{s}) \to \mathbb{L}$. This, together with Lemma 14.3, can be easily seen to imply (24) by LC. ■

14.6 Q.E.D.

In this subsection we finish the extensional completeness proof for **CLA4**. Well, almost finish. The point is that our argument relies on Lemma 14.6 whose proof is postponed to Appendix A. Here we only present a brief intuitive explanation of the proof idea for it. A reader satisfied by our explanation will have no reasons to go through the technical appendix given at the end of this paper, whose only purpose is to provide a relatively detailed proof for Lemma 14.6.

Let E be a formula not containing the variable y. We say that a formula H is a $(\bot, y)$**-development** of E iff H is the result of replacing in E:

- either a surface occurrence of a subformula $F_0 \sqcap F_1$ by F_i ($i = 0$ or $i = 1$),
- or a surface occurrence of a subformula $\sqcap x F(x)$ by $F(y)$.

$(\top, y)$**-development** is defined in the same way, only with $\sqcup, \bigsqcup$ instead of $\sqcap, \bigsqcap$.

Lemma 14.6 *Assume $E(\vec{s})$ is a formula all of whose free variables are among $\vec{s}$, and y is a variable not occurring in $E(\vec{s})$. Then:*

(a) For every $(\bot, y)$-development $H_i(y, \vec{s})$ of $E(\vec{s})$, **CLA4** *proves $E^\circ_\circ(z, \vec{s}) \to \bigsqcup u H^\circ_i(u, y, \vec{s})$.*

(b) Where $H_1(y, \vec{s}), \ldots, H_n(y, \vec{s})$ are all of the $(\top, y)$-developments of $E(\vec{s})$, **CLA4** *proves*

$$E^\circ(z, \vec{s}) \to E^\circ_\circ(z, \vec{s}) \sqcup \mathbb{L} \sqcup \bigsqcup u \bigsqcup y H^\circ_1(u, y, \vec{s}) \sqcup \ldots \sqcup \bigsqcup u \bigsqcup y H^\circ_n(u, y, \vec{s}). \tag{25}$$

Proof idea. $E^\circ(z, \vec{s})$ implies that z is a configuration reached by $\mathcal{X}$ in some play, and the game by that time has been brought down to $E(\vec{s})$. $E^\circ_\circ(z, \vec{s})$ additionally implies that this situation persists "for a while" after z.

For clause (a), assume $E^\circ_\circ(z, \vec{s})$. For any $(\bot, y)$-development $H_i(y, \vec{s})$ of $E(\vec{s})$ and any value of y, $H_i(y, \vec{s})$ is the game to which $E(\vec{s})$ is brought down by a certain labmove $\bot\alpha$. To solve $\bigsqcup u H^\circ_i(u, y, \vec{s})$ — i.e., make

$H_i^\circ(u, y, \vec{s})$ true — we can choose u to be the result of appending such a labmove $\bot\alpha$ to the run tape content of the deterministic successor of configuration z. After properly formalizing encoding for configurations, this argument can be reproduced in **CLA4**.

For clause (b), assume $E^\circ(z, \vec{s})$. We can trace, within **CLA4**, the work of $\mathcal{X}$ for "sufficiently many" — namely, $\chi(|z|)$ — steps in the scenario where Environment does not move. If $\mathcal{X}$ does not move during those $\chi(|z|)$ steps either, then $E_\circ^\circ(z, \vec{s})$ is true and we can choose it in the consequent of (25). Suppose now $\mathcal{X}$ makes a move α within $\chi(|z|)$ steps. If α is illegal, then $\mathbb{L}$ is true, and we choose the latter in the consequent of (25). Otherwise, if α is a legal move, then it brings $E(\vec{s})$ down to one of (the instances of) its $(\top, y)$-developments $H_i(y, \vec{s})$ for a certain value of y. We choose the corresponding $\sqcup$-disjunct in the consequent of (25); further, by tracing the work of $\mathcal{X}$, we will be able to compute the value of the configuration u in which $\mathcal{X}$ made the above move, as well the above-mentioned "certain value of y". By choosing these values for the variables u and y in $\sqcup u \sqcup y H_1^\circ(u, y, \vec{s})$, we win the game. Again, after properly formalizing encoding for configurations, the entire argument can be reproduced in **CLA4**. ■

Lemma 14.7 *Assume $E(\vec{s})$ is a formula all of whose free variables are among $\vec{s}$. Then* **CLA4** *proves* $E^\circ(z, \vec{s}) \to \overline{E(\vec{s})}$.

Proof. We prove this lemma by (meta)induction on the complexity of $E(\vec{s})$. By the induction hypothesis, for any $(\bot, y)$- or $(\top, y)$-development $H_i(y, \vec{s})$ of $E(\vec{s})$ (if there are any), **CLA4** proves

$$H_i^\circ(u, y, \vec{s}) \to \overline{H_i(y, \vec{s})}. \tag{26}$$

Argue in **CLA4** to justify $E^\circ(z, \vec{s}) \to \overline{E(\vec{s})}$. Consider any values (constants) b and $\vec{a}$ chosen by Environment for z and $\vec{s}$, respectively.[12] Assume $E^\circ(b, \vec{a})$ is true (if not, we win). Our goal is to show how to win $\overline{E(\vec{a})}$. From clause (b) of Lemma 14.6, the resource (25) with $b, \vec{a}$ plugged for $z, \vec{s}$ is at our disposal. Since the antecedent of the latter is true, the provider of (25) will have to choose one of the $\sqcup$-disjuncts in the consequent

$$E_\circ^\circ(b, \vec{a}) \sqcup \mathbb{L} \sqcup \sqcup u \sqcup y H_1^\circ(u, y, \vec{a}) \sqcup \ldots \sqcup \sqcup u \sqcup y H_n^\circ(u, y, \vec{a}). \tag{27}$$

Case 1: $\mathbb{L}$ is chosen in (27). It has to be true, or else the provider loses. By Lemma 14.3, we also have the resource $\mathbb{L} \to \forall \overline{E(\vec{s})}$. Since its antecedent $\mathbb{L}$ is true, we know how to win the consequent $\forall \overline{E(\vec{s})}$. But a strategy that wins the latter, of course, also wins our target $\overline{E(\vec{a})}$.

Case 2: One of $\sqcup u \sqcup y H_i^\circ(u, y, \vec{a})$ is chosen in (27). This should be followed by a further choice of some constants c and d for u and y, yielding $H_i^\circ(c, d, \vec{a})$. Plugging $\vec{a}$, c and d for $\vec{s}$, u and y in (26), we get $H_i^\circ(c, d, \vec{a}) \to \overline{H_i(d, \vec{a})}$. We may assume that the antecedent of the latter is true, or else the provider of (27) lied when bringing the latter down to $H_i^\circ(c, d, \vec{a})$. Thus, the consequental resource $\overline{H_i(d, \vec{a})}$ is at our disposal. But, remembering that the formula $H_i(y, \vec{s})$ is a $(\top, y)$-development of the formula $E(\vec{s})$, we can now win $\overline{E(\vec{a})}$ by making a move α that brings the latter down to $\overline{H_i(d, \vec{a})}$, which we already know how to win. For example, imagine $E(\vec{s})$ is $Y(\vec{s}) \to Z(\vec{s}) \sqcup T(\vec{s})$ and $H_i(y, \vec{s})$ is $Y(\vec{s}) \to Z(\vec{s})$, so that $\overline{E(\vec{a})}$ is $\overline{Y(\vec{a}) \to Z(\vec{a})} \sqcup \overline{T(\vec{a})}$ and $\overline{H_i(d, \vec{a})}$ is $\overline{Y(\vec{a}) \to Z(\vec{a})}$. Then the above move α will be "1.0". It indeed brings $\overline{Y(\vec{a}) \to Z(\vec{a})} \sqcup \overline{T(\vec{a})}$ down to $\overline{Y(\vec{a}) \to Z(\vec{a})}$. As another example, imagine $E(\vec{s})$ is $Y(\vec{s}) \to \sqcup w Z(w, \vec{s})$ and $H_i(y, \vec{s})$ is $Y(\vec{s}) \to Z(y, \vec{s})$, so that $\overline{E(\vec{a})}$ is $\overline{Y(\vec{a}) \to \sqcup w Z(w, \vec{a})}$ and $\overline{H_i(d, \vec{a})}$ is $\overline{Y(\vec{a}) \to Z(d, \vec{a})}$. Then the above move α will be "1.d". It indeed brings $\overline{Y(\vec{a}) \to \sqcup w Z(w, \vec{a})}$ down to $\overline{Y(\vec{a}) \to Z(d, \vec{a})}$.

Case 3: $E_\circ^\circ(b, \vec{a})$ is chosen in (27). It has to be true, or else the provider loses.

Subcase 3.1: The formula $E(\vec{s})$ is critical. Since $E_\circ^\circ(b, \vec{a})$ is true, so is $\exists E_\circ^\circ(z, \vec{s})$. By Lemma 14.5, we also have $\exists E_\circ^\circ(z, \vec{s}) \to \forall \overline{E(\vec{s})}$. So, we have a strategy that wins $\forall \overline{E(\vec{s})}$. Of course, the same strategy also wins $\overline{E(\vec{a})}$.

Subcase 3.2: The formula $E(\vec{s})$ is not critical. By Lemma 14.4, we find that the elementarization of $\overline{E(\vec{a})}$ is true. This obviously means that if Environment does not move in $\overline{E(\vec{a})}$, we win the latter. So, assume Environment makes a move α in $\overline{E(\vec{a})}$. The move should be legal, or else we win. Of course, the same move is a legal move of $E(\vec{a})$ and, for one of the $(\bot, y)$-developments $H_i(y, \vec{s})$ of the formula $E(\vec{s})$ and some constant c, it brings $E(\vec{a})$ down to $H_i(c, \vec{a})$ as well as $\overline{E(\vec{a})}$ down to $\overline{H_i(c, \vec{a})}$. For example, if $E(\vec{s})$ is $Y(\vec{s}) \to Z(\vec{s}) \sqcap T(\vec{s})$, α could be the move "1.0", which brings $Y(\vec{a}) \to Z(\vec{a}) \sqcap T(\vec{a})$ down to $Y(\vec{a}) \to Z(\vec{a})$; the formula $Y(\vec{s}) \to Z(\vec{s})$ is indeed a $(\bot, y)$-development of the formula $Y(\vec{s}) \to Z(\vec{s}) \sqcap T(\vec{s})$.

[12] Here, unlike the earlier followed practice, for safety, we are reluctant to use the names z and $\vec{s}$ for those constants.

As another example, imagine $E(\vec{s})$ is $Y(\vec{s}) \to \sqcap w Z(w, \vec{s})$. Then the above move α could be "1.c", which brings $Y(\vec{a}) \to \sqcap w Z(w, \vec{a})$ down to $Y(\vec{a}) \to Z(c, \vec{a})$; the formula $Y(\vec{s}) \to Z(y, \vec{s})$ is indeed a $(\bot, y)$-development of the formula $Y(\vec{s}) \to \sqcap w Z(w, \vec{s})$. Fix the above formula $H_i(y, \vec{s})$ and constant c. Choosing b, $\vec{a}$ and c for z, $\vec{s}$ and y in the resource provided by clause (a) of Lemma 14.6, we get the resource $E^\circ_\circ(b, \vec{a}) \to \sqcup u H^\circ_i(u, c, \vec{a})$. Since the antecedent of the latter is true by our assumptions, the consequent $\sqcup u H^\circ_i(u, c, \vec{a})$ is at our disposal. The provider will have to choose a constant d for u in it such that $H^\circ_i(d, c, \vec{a})$ is true. Hence, by choosing d, c and $\vec{a}$ for u, y and $\vec{s}$ in (26), we get the resource $\overline{H_i(c, \vec{a})}$. That is, we have a strategy for the game $\overline{H_i(c, \vec{a})}$ to which $\overline{E(\vec{a})}$ has evolved after Environment's move α. We switch to that strategy and win. ■

Now we are ready to claim the target result of this section. Let a be the code of the start configuration of $\mathcal{X}$, and $\hat{a}$ be a standard variable-free term representing a, such as 0 followed by a "$'$"s. Of course, **PA** and hence **CLA4** proves $X^\circ(\hat{a})$. By Fact 12.6, **CLA4** proves $\sqcup z(z{=}\hat{a})$. By Lemma 14.7, **CLA4** also proves $\sqcap z\big(X^\circ(z) \to \overline{X}\big)$. These three can be seen to imply $\overline{X}$ by LC. Thus, **CLA4** $\vdash \overline{X}$, as desired.

15 Inherent extensional incompleteness in the general case

The extensional completeness of **CLA4** is not a result that could be taken for granted. In this short section we argue that, if one replaces *polynomial time computability* by simply *computability* in our semantical treatment of **CLA4**, extensional completeness is impossible to achieve for whatever recursively axiomatizable sound extension of **CLA4** or other systems of clarithmetic.

Our extensional incompleteness argument goes like this. Consider any system **S** in the style of **CLA4** whose proof predicate — throughout this section understood in the "superextended" sense of Section 11 — is decidable and hence the theoremhood predicate is recursively enumerable. Assume **S** is sound in the same strong sense as **CLA4** — that is, there is an effective procedure that extracts an algorithmic solution (HPM) for the problem represented by any sentence F from any **S**-proof of F.

Let then $A(s)$ be the predicate which is true iff:

- s is (the code of) an **S**-proof of some sentence of the form $\sqcap x\big(\neg E(x) \sqcup E(x)\big)$, where E is elementary,
- and $E(s)$ is false.

On our assumption of the soundness of **S**, $A(s)$ is a decidable predicate. Namely, it is decided by a procedure that first checks if s is the code of an **S**-proof of some sentence of the form $\sqcap x\big(\neg E(x) \sqcup E(x)\big)$, where E is elementary. If not, it rejects. If yes, the procedure extracts from s an HPM $\mathcal{H}$ which solves $\sqcap x\big(\neg E(x) \sqcup E(x)\big)$, and then simulates the play of $\mathcal{H}$ in the scenario where, at the very beginning of the play, Environments makes the move s, thus bringing the game down to $\neg E(s) \sqcup E(s)$. If, in this play, $\mathcal{H}$ responds by choosing $\neg E(s)$, then the procedure accepts s; and if $\mathcal{H}$ responds by choosing $E(s)$, then the procedure rejects s. Obviously this procedure indeed decides the predicate A.

Now, assume that **S** is extensionally complete. Since A is decidable, the problem $\sqcap x\big(\neg A(x) \sqcup A(x)\big)$ has an algorithmic solution. So, for some sentence F with $F^\dagger = \sqcap x\big(\neg A(x) \sqcup A(x)\big)$ and some c, we should have that c is (the code of) an **S**-proof of F. Obviously F should have the form $\sqcap x\big(\neg E(x) \sqcup E(x)\big)$, where $E(x)$ is an elementary formula with $E^\dagger(x) = A(x)$. We are now dealing with the absurd of $A(c)$ being true iff it is false.

16 On the intensional strength of CLA4

Theorem 16.1 *Let X and $\mathbb{L}$ be as in Section 14. Then* **CLA4** $\vdash \neg\mathbb{L} \to X$.

Proof. Let X and $\mathbb{L}$ be as in Section 14, and so be the meaning of the overline notation. First, by induction on the complexity of E, we want to show that

$$\textit{For any formula } E, \ \mathbf{CLA4} \vdash \forall(\overline{E} \wedge \neg\mathbb{L} \to E). \tag{28}$$

If E is a literal, then $\forall(\overline{X} \wedge \neg\mathbb{L} \to E)$ is nothing but $\forall\big((E \vee \mathbb{L}) \wedge \neg\mathbb{L} \to E\big)$. This is a classically valid elementary sentence, and hence it is provable in **CLA4** (by LC from the empty set of premises). Next, suppose E

is $F_0 \wedge F_1$. By the induction hypothesis, **CLA4** proves both $\forall(\overline{F_0} \wedge \neg\mathbb{L} \to F_0)$ and $\forall(\overline{F_1} \wedge \neg\mathbb{L} \to F_1)$. These two, by LC, imply $\forall\big((\overline{F_0} \wedge \overline{F_1}) \wedge \neg\mathbb{L} \to F_0 \wedge F_1\big)$. And the latter is nothing but the desired $\forall(\overline{E} \wedge \neg\mathbb{L} \to E)$. The remaining cases where E is $F_0 \vee F_1$, $F_0 \sqcap F_1$, $F_0 \sqcup F_1$, $\sqcap xF(x)$, $\sqcup xF(x)$, $\forall xF(x)$ or $\exists xF(x)$ are handled in a similar way. (28) is thus proven.

(28) implies that **CLA4** proves $\overline{X} \wedge \neg\mathbb{L} \to X$. As established in Section 14, **CLA4** also proves $\overline{X}$. From these two, by LC, **CLA4** proves $\neg\mathbb{L} \to X$, as desired. ■

Remember that, in Section 14, X was an arbitrary **CLA4**-sentence assumed to have a polynomial time solution under the standard interpretation †. And $\neg\mathbb{L}$ was a certain true sentence of the language of classical Peano arithmetic. We showed in that section that **CLA4** proved a certain sentence $\overline{X}$ with $\overline{X}^\dagger = X^\dagger$. That is, we showed that X was "extensionally provable". According to our present Theorem 16.1, in order to make X also provable in the intensional sense, all we need is to add to the axioms of **CLA4** the true elementary sentence $\neg\mathbb{L}$.

In philosophical terms, the import of Theorem 16.1 is that the culprit of the intensional incompleteness of **CLA4** is the (Gödel's) incompleteness of its classical, elementary part. Otherwise, the "nonelementary rest" of **CLA4** — the two extra-Peano axioms and the **CLA4**-Induction rule — as a bridge from classical arithmetic to polynomial-time-computability-oriented clarithmetic, is complete in a certain very strong and natural sense. Namely, it guarantees not only extensional but also intensional provability of every polynomial time computable problem as long as all necessary true elementary sentences are taken care of. This means that if, instead of **PA**, we take the truth arithmetic **Th(N)** (the set of all true sentences of the language of **PA**) as the base arithmetical theory, the corresponding version of **CLA4** will be not only extensionally, but also intensionally complete. Unfortunately, however, such a system will no longer be recursively axiomatizable.

To summarize, in order to make **CLA4** intensionally stronger, it would be sufficient to add to it new true elementary (classical) sentences only. Note that this sort of an extension, even if in a language more expressive than that of **PA**, would automatically remain sound and extensionally complete: virtually nothing in this paper relies on the fact that **PA** is not stronger than it really is. Thus, basing applied theories on CoL allows us to construct ever more expressive and intensionally strong theories without worrying about how to preserve soundness and extensional completeness. Among the main goals of this paper was to illustrate the scalability of CoL rather than the virtues of the particular system **CLA4** based on it. The latter is in a sense arbitrary, as is **PA** itself: in the role of the classical part of **CLA4**, we could have chosen not only any true extension of **PA**, but certain weaker-than-**PA** theories as well, for our proof of the extensional completeness of **CLA4** does not require the full strength of **PA**. The reason for not having done so is purely "pedagogical": **PA** is the simplest and best known arithmetical theory, and reasoning in it is much more relaxed, easy and safe than in weaker versions. **CLA4** is thus the simplest and nicest representative of the wide class of clarithmetical theories for polynomial time computability, all enjoying the same relevant properties as **CLA4** does.

As pointed out in Section 1, among the potential applications of **CLA4**-style systems is using them as formal tools (say, after developing reasonable theorem-provers) for systematically finding efficient solutions for problems, and the stronger such a system is, the better the chances that a solution for a given problem will be found. Of course, what matters in this context is intensional rather than extensional strength. So, perfect strength is not achievable, but we can keep moving ever closer to it.

One may ask why not think of simply using **PA** (or even, say, **ZFC**) instead of **CLA4** for the same purposes: after all, **PA** is strong enough to allow us reason about polynomial time computability. This is true, but **PA** is far from being a reasonable alternative to **CLA4**. First of all, as a tool for finding solutions, **PA** is very indirect and hence hopelessly inefficient. Pick any of the basic functions of Section 12 and try to generate, in **PA**, a full formal proof of the fact that the function is polynomial-time computable (or even just *express* this fact) to understand the difference. Such a proof would have to proceed by clumsy reasoning about *non-number* objects such as Turing machines and computations, which, only by good luck, happen to be amenable to being understood as numbers through encoding. In contrast, reasoning in **CLA4** is directly about numbers and their properties, without having to encode any foreign (metaarithmetical or complexity-theoretical) beasts and then try to reason about them as if they were just kind and innocent natural numbers. Secondly, even if an unimaginably strong theorem-prover succeeded in finding such a proof, there would be no direct use of it because, from a proof of the existence of a solution we cannot directly extract a solution.

Furthermore, even knowing that a given HPM $\mathcal{X}$ solves the problem in *some* polynomial time χ, would have no practical significance without knowing *what* particular polynomial χ is, in order to assess whether it is reasonable for our purposes, or takes us beyond the number of nanoseconds in the lifespan of the universe (after all, $\ell^{999^{999}}$ is also a polynomial function!). In order to actually obtain a solution and a polynomial bound for it, one would need a **constructive** proof, that is, not just a proof that a polynomial function χ and a χ-time solution exist, but a proof of the fact that certain particular numbers a and b are (the codes of) a polynomial term χ and a χ-time solution $\mathcal{X}$. Otherwise, a theorem-prover would have to be used not just once for a single target formula, but an indefinite (intractably many) number of times, once per each possible pair of values of a, b until the "right" values are encountered. To summarize, **PA** does not provide any reasonable mechanism for handling queries in the style "*find* a polynomial time solution for problem A": in its standard form, **PA** is merely a YES/NO kind of a "device".

The above dark picture can be somewhat brightened by switching from **PA** to Heyting's arithmetic **HA** — the version of **PA** based on intuitionistic logic instead of classical logic, which is known to allow us to directly extract, from a proof of a formula $\exists x F(x)$, a particular value of x for which $F(x)$ is true. But the question is why intuitionistic logic and not CoL? Both claim to be "constructive logics", but the constructivistic claims of CoL have a clear semantical meaning and justification, while intuitionistic logic is essentially an ad hoc invention whose constructivistic claims are mainly based on certain syntactic and hence circular considerations,[13] without being supported by a convincing and complete *formal* constructive semantics. And, while **HA** is immune to the second one of the two problems pointed out in the previous paragraph, it still suffers from the first problem. At the same time, as a reasoning tool, **HA** is inferior to **PA**, for it is intensionally weaker and, from the point of view of the philosophy of CoL, is so for no good reasons. As a simple example, consider the function f defined by "$f(x) = x$ if **PA** is either consistent or inconsistent, and $f(x) = 2x$ otherwise". This is a legitimately defined function, and we all — just as **PA** — know that extensionally it is the same as the identity function $f(x) = x$. Yet, **HA** can be seen to fail to prove — in the intensional sense — its computability. Despite its name, intuitionistic logic is not so "intuitive" after all!

A natural question to ask is: *Is there a sentence X of the language of* **CLA4** *whose polynomial time computability is constructively provable in* **PA** *yet X is not provable in* **CLA4***?* Remember that, as we agreed just a while ago, by **constructive provability** of the polynomial time computability of X in **PA** we mean that, for some particular HPM $\mathcal{X}$ and a particular polynomial (term) χ, **PA** proves that $\mathcal{X}$ is a χ-time solution of X. If the answer to this question was positive, then **PA**, while indirect and inefficient, would still have at least *something* to say in its defense when competing with **CLA4** as a problem-solving tool. But, as seen from the following theorem, the answer to the question is negative:

Theorem 16.2 *Let X be any sentence of the language of* **CLA4** *such that* **PA** *constructively proves (in the above sense) the polynomial time computability of X. Then* **CLA4** $\vdash X$.

Proof. Consider any sentence X of the language of **CLA4**. Assume **PA** constructively proves the polynomial time computability of X, meaning that, for a certain HPM $\mathcal{X}$ and a certain term χ, **PA** proves that $\mathcal{X}$ solves X in time χ. But this is exactly what the sentence $\mathbb{L}$ of Section 14 denies. So, **PA** $\vdash \neg\mathbb{L}$. But, by Theorem 16.1, we also have **CLA4** $\vdash \neg\mathbb{L} \to X$. Consequently, **CLA4** $\vdash X$. ■

An import of the above theorem is that, if we tried to add to **CLA4** some new nonelementary axioms in order to achieve a properly greater intensional strength, the fact that such axioms are computable in time χ for some particular polynomial χ would have to be unprovable in **PA**, and hence would have to be "very nontrivial". The same applies to attempts to extend **CLA4** through some new rules of inference.

17 Give Caesar what belongs to Caesar

Beginning from Buss's seminal work [6], many complexity-sensitive or complexity-oriented arithmetical and logical systems have been developed by various authors ([3, 7, 8, 9, 10, 33, 35] and more). Most of those

[13] What creates circularity is the common-sense fact that syntax is merely to serve a meaningful semantics, rather than vice versa. It is hard not to remember the following words from [26] here: "The reason for the failure of $P \sqcup \neg P$ in CoL is not that this principle ... is not included in its axioms. Rather, the failure of this principle is exactly the reason why this principle, or anything else entailing it, would not be among the axioms of a sound system for CoL".

achieve control over complexity in ways very different from ours, such as by type information rather than by explicit bounds on quantifiers, for which reason we do not attempt any direct comparison here. As pointed out in Section 1, Buss's [6] original work on bounded arithmetic is closest to — and the most immediate precursor of — our approach. In fact, in a broad sense, **CLA4** *is* a system of bounded arithmetic, only based on CoL instead of classical logic or intuitionistic logic on which the other systems of bounded arithmetic have been traditionally based.

The main relevant results in the studies of classical-logic-based bounded arithmetic, extensive surveys of which can be found in [11, 32], can be summarized saying that, by appropriately weakening the induction axiom of **PA** and restricting it to bounded formulas of certain forms, and correspondingly readjusting the nonlogical vocabulary and axioms of **PA**, certain soundness and completeness for the resulting system(s) **S** can be achieved. Such soundness results typically read like "If **S** proves a formula of the form $\forall x \exists y F(x, y)$, where F satisfies such and such constraints, then there is function of such and such computational complexity which, for each a, returns a b with $F(a, b)$". And completeness results typically read like "For any function f of such and such computational complexity, there is an **S**-provable formula of the form $\forall x \exists y F(x, y)$ such that, for any a and b, $F(a, b)$ is true iff $b = f(a)$".

Among the characteristics that make our approach very different from the above (as well as any other complexity-oriented systems of arithmetic known to the author), one should point out that it *extends* rather than *restricts* the language and the deductive power of **PA**. Restricting **PA** can be seen as throwing out the baby with the bath water. Not only does it expel from the system many complexity-theoretically unsound yet otherwise meaningful and useful theorems, but it also reduces — even if only in the intensional rather than extensional sense — the class of complexity-theoretically correct provable principles. This is a necessary sacrifice, related to the inability of the underlying classical logic to clearly differentiate between constructive ($\sqcap$, $\sqcup$, $\bigsqcap$, $\bigsqcup$) and "ordinary", non-constructive versions ($\wedge$, $\vee$, $\forall$, $\exists$) of operators. The inadequacy of classical logic as a basis for constructive systems also shows itself in the fact that the above-mentioned soundness and completeness results are only partial.

The above problem of partiality is partially overcome when one bases a complexity-oriented arithmetic on intuitionistic logic ([7, 35]) instead of classical logic. In this case, soundness/completeness extends to all formulas of the form $\forall x \exists y F(x, y)$, without the "$F$ satisfies such and such constraints" condition (the reason why we still consider this sort of soundness/completeness partial is that it remains limited to formulas of the form $\forall x \exists y F(x, y)$, i.e. functions, which, for us, are only special cases of computational problems). However, for reasons pointed out in the previous section, switching to intuitionistic logic signifies throwing out even more of the "baby" from the bath tub, further decreasing the intensional strength of the theory and probably losing its intuitive clarity or appeal in the eyes of the classically-minded majority.

Both classical-logic-based and intuitionistic-logic-based systems of bounded arithmetic happen to be *inherently weak* theories, as opposed to our CoL-based version, which is as strong as Gödel's incompleteness phenomenon permits, and which can be indefinitely strengthened without losing computational soundness. We owe this achievement to the fact that CoL gives Caesar what belongs to Caesar, and God what belongs to God. As we had a chance to see throughout this paper, classical ($\wedge$, $\vee$, $\forall$, $\exists$) and constructive ($\sqcap$, $\sqcup$, $\bigsqcap$, $\bigsqcup$) logical constructs can peacefully coexist and complement each other in one natural system that seamlessly extends the classical, constructive, resource- and complexity-conscious visions and concepts, and does so not by mechanically putting things together, but rather on the basis of one natural, all-unifying, complete game semantics. Unlike most other approaches where only few, special-form expressions (if any) have clear computational interpretations, in our case every formula is a meaningful computational problem. Further, we can capture not only computational problems in the traditional sense, but also problems in the more general — interactive — sense.

Classical logic and classical arithmetic, so close (unlike, say, intuitionistic logic or **HA**) to the heart and mind of all of us, do not at all need to be rejected or tampered with in order to achieve constructive heights. Just the opposite, they can be put in faithful and useful service to this noble goal. Our heavy reliance on reasoning in **PA** throughout this paper is an eloquent illustration of it. Overall, the present work can be seen as an illustration of the fruitfulness of two independently conceived lines of thought — bounded arithmetic and computability logic — through a successful marriage between them.

The forthcoming paper [30] constructs three new, incrementally strong systems of clarithmetic, named **CLA5**, **CLA6** and **CLA7**. In the same sense as **CLA4** is sound and complete with respect to polynomial time computability, these systems are shown to be sound and complete with respect to polynomial space

computability, elementary recursive time computability and primitive recursive time computability, respectively (as for elementary recursive space and primitive recursive space, they simply coincide with elementary recursive time and primitive recursive time). The simplicity and elegance of those systems serves as additional empirical evidence for the naturalness of basing applied theories on CoL instead of the more traditional alternatives, and for the flexibility and scalability of our approach. All three systems, on top of Axioms 1-7, have Axiom 8 as the only extra-Peano axiom (Axiom 9 simply becomes derivable and hence redundant due to the presence of a stronger induction rule). **CL12** continues to serve as the logical basis for these systems, and what varies is only the induction rule. The induction rule of **CLA5** differs from that of **CLA4** in that, while the (two) inductive steps of the latter are based on binary successors, the (single) inductive step of the former is based on unary successor, i.e., is the kind old $F(x) \to F(x')$, with $F(x)$ still required to be a polynomially bounded formula. The system **CLA6** is obtained from **CLA5** by relaxing this requirement in the induction rule and, instead, requiring that $F(x)$ be exponentially bounded. And the system **CLA7** is obtained from **CLA6** by removing all conditions on $F(x)$ whatsoever, thus leaving the realm of bounded arithmetic.

The earlier mentioned system **CLA1** of [27] further strengthens the above series. Its logical basis, just like that of all clarithmetical theories we have seen, is **CL12**. And the nonlogical axioms, just as in the case of **CLA5**, **CLA6** and **CLA7**, are Axioms 1-8 of Section 11. As we may guess, the only difference between **CLA1** and the weaker systems **CLA4**-**CLA7** is (again) related to how the induction rule operates. Here the difference is of a qualitative character due to the fact that **CLA1**, unlike **CLA4**-**CLA7**, is a *natural deduction* system. Namely, while an inductive step of **CLA7** is a *formula* $F(x) \to F(x')$, the corresponding inductive step in **CLA1** is a *derivation* of $F(x')$ from $F(x)$. In classical systems, according to the deduction theorem, a formula E is derivable from a formula G iff the formula $G \to F$ is provable, so switching to natural deduction in the style of **CLA1** would create no difference. The situation, however, is very different in (the resource-conscious) CoL-based systems, where deriving E from G is generally easier than proving $G \to E$. This is so because a derivation may "recycle" its premises while, on the other hand, the antecedent of a $\to$-combination may be "unrecyclable". For instance, $E \wedge E$ is always derivable from E but, as we had a chance to see from Exercise 9.5, $E \to E \wedge E$ is not always provable (and/or valid). While derivability of $F(x')$ from $F(x)$ thus does not generally imply provability of $F(x) \to F(x')$, the latter *does* always imply the former. Consequently, **CLA1** is at least as strong as **CLA7**, meaning that **CLA1**, just like **CLA7**, can extensionally prove all primitive recursive time (and/or space) computable problems. A natural expectation here is that, at the same time, **CLA1** takes us "far beyond" primitive recursive time (and/or space) computability, even though exactly *how far* still remains to be understood.

A Appendix

Throughout the rest of this appendix, the sole purpose of which is to prove Lemma 14.6, X, $\mathcal{X}$ and χ are as in Section 14. The terms "configuration", "state", "tape symbol" etc. exclusively refer to ones of $\mathcal{X}$. We assume that 0 and 1 are among the tape symbols. **blank** will stand for the blank tape symbol.

The proofs given in this appendix will heavily and repeatedly rely on **PA** and the results of Section 12. It is important to note that, almost always, this reliance will be only *implicit*.

A.1 Encoding configurations

In order to prove Lemma 14.6, we need to introduce a system of encoding for various objects of relevance. Whenever O is such an object, $\ulcorner O \urcorner$ will stand for its code.

Let $\mathcal{A}$ be the set consisting of all states (of $\mathcal{X}$), and four versions $\hat{a}$, $\check{a}$, $\underline{\hat{a}}$, $\underline{\check{a}}$ of every tape symbol a. As opposed to *tape symbols*, we refer to the elements of $\mathcal{A}$ (simply) as **symbols**. As we are going to see shortly, in our encoding of configurations, $\hat{a}$ (resp. $\check{a}$) means the tape symbol a written on the work (resp. run) tape, and the presence (resp. absence) of an underline indicates that the head of the corresponding tape is (resp. is not) currently looking at the cell containing a.

We extend the $\hat{}$, $\check{}$ notation from tape symbols to strings of tape symbols. Namely, for any such string α, $\hat{\alpha}$ means the result of replacing every symbol a by $\hat{a}$ in α. Similarly for $\check{\alpha}$.

We pick and fix a sufficiently large integer $\mathfrak{k}$ and, with $\mathfrak{K}$ standing for $2^{\mathfrak{k}}$ throughout the rest of this paper, encode each symbol a as a natural number $\ulcorner a \urcorner$ with $|\ulcorner a \urcorner| = \mathfrak{K}$. As practiced earlier, terminologically and

notationally we identify such an $\ulcorner a \urcorner$ with the corresponding binary numeral. Thus, the codes of symbols are bit strings, all of length $\mathfrak{K}$ and none starting with a 0. Needless to mention that different symbols are required to have different codes.

Further, where $a_1, \ldots, a_k$ is a sequence of symbols, we encode it as the binary numeral — again, identified with the corresponding number — $\ulcorner a_1 \urcorner \ldots \ulcorner a_k \urcorner$. We will not always be careful about differentiating objects from their codes, and may say something like "the symbol b" where, strictly speaking, the code of that symbol is meant, or vice versa.

We need to make clear what, exactly, is meant by a configuration. According to our earlier informal explanation, this is a full description of some "current" situation in $\mathcal{X}$, namely, a list indicating the state of $\mathcal{X}$, the locations of its two scanning heads, and the contents of its two tapes. The tapes, however, are infinite, and we need to agree on how to represent their contents by finite means. Remember our convention that a head of an HPM can never move past the leftmost blank cell of the corresponding tape, and that the work-tape head can never write the blank symbol. This means that every cell to the left of a blank cell will also be blank and, accordingly, when describing a configuration, it would be sufficient to describe the contents of its tapes up to (including) the leftmost blank cells. Precisely, we agree to understand each configuration C as the following sequence of symbols:

$$a,\ \hat{b}_0,\ \ldots\ \hat{b}_{i-1},\ \underline{\hat{b}}_i,\ \hat{b}_{i+1},\ \ldots,\ \hat{b}_m,\ \ \check{c}_0,\ \ldots\ \check{c}_{j-1},\ \underline{\check{c}}_j,\ \check{c}_{j+1},\ \ldots\ \check{c}_n \tag{29}$$

where, in the context of C, a is the ("current") state of $\mathcal{X}$, $b_0 \ldots b_m$ (resp. $c_0 \ldots c_n$) are the contents of cells #0 through #m (resp. #n) of the work (resp. run) tape, and i (resp. j) is the cell # of the cell scanned by the head of the work (resp. run) tape. In addition, both $b_m = c_n =$ **blank** while no other b_k or c_k is **blank**. We encode the above configuration as any other sequence of symbols, i.e., as the binary numeral

$$\ulcorner a \urcorner \ulcorner \hat{b}_0 \urcorner\ \ldots\ \ulcorner \hat{b}_{i-1} \urcorner \ulcorner \underline{\hat{b}}_i \urcorner \ulcorner \hat{b}_{i+1} \urcorner\ \ldots\ \ulcorner \hat{b}_m \urcorner\ \ulcorner \check{c}_0 \urcorner\ \ldots\ \ulcorner \check{c}_{j-1} \urcorner \ulcorner \underline{\check{c}}_j \urcorner \ulcorner \check{c}_{j+1} \urcorner\ \ldots\ \ulcorner \check{c}_n \urcorner.$$

In the sequel, we will be using the pseudoterm $x \circ y$ and several elementary formulas with special names, each one being a natural arithmetization of the corresponding predicate shown below:

- $x \circ y$ abbreviates $x \times 2^{|y|} + y$. Note that, when x and y are the codes of some sequences $a_1, \ldots, a_m$ and $b_1, \ldots, b_n$ of symbols, $x \circ y$ is the code of the **concatenation** $a_1, \ldots, a_m, b_1, \ldots, b_n$ of those sequences.
- $\mathbb{N}(x, y)$ says "if x is $\ulcorner \hat{b}_1, \ldots, \hat{b}_k \urcorner$ for some symbols $b_1, \ldots, b_k$ ($k \geq 0$), then y is $\ulcorner \check{b}_1, \ldots, \check{b}_k \urcorner$." So, for instance, $\mathbb{D}(\ulcorner \hat{1}, \hat{\cdot}, \hat{0} \urcorner, \ulcorner \check{1}, \check{\cdot}, \check{0} \urcorner)$ is true.
- $\mathbb{C}(x)$ says "x is the code of a configuration".
- $\mathbb{I}(x, y)$ says "x is the code of a configuration of the form (29), and $i = y$".
- $\mathbb{J}(x, y)$ says "x is the code of a configuration of the form (29), and $j = y$".
- $\mathbb{M}(x, y)$ says "x is the code of a configuration of the form (29), and $m = y$".
- $\mathbb{E}(x, y)$ says "y is the code of the sequence of symbols resulting from changing every 0 to $\check{0}$ and every 1 to $\check{1}$ in the binary numeral representing number x". So, for instance, $\mathbb{E}(101, \ulcorner \check{1}, \check{0}, \check{1} \urcorner)$ is true.
- $\mathbb{D}(x, y)$ says "x is $\ulcorner \hat{b}_1, \ldots, \hat{b}_k \urcorner$ for some bits $b_1, \ldots, b_k$ ($k \geq 0$) where b_1 (if present) is 1, and y is the number represented by the numeral $b_1 \ldots b_k$". So, for instance, $\mathbb{D}(\ulcorner \hat{1}, \hat{0}, \hat{1} \urcorner, 101)$ is true.
- $\mathbb{S}(x, y)$ says "x is the code of a configuration and y is the code of the deterministic successor (see Section 14.5) of that configuration".
- $\mathbb{A}(z, x, y)$ says "z is the code of a legitimate configuration C (see Section 14.5), x is the code of the yth deterministic successor of C, and, for any i with $0 \leq i \leq y$, the state of the ith deterministic successor of C is not a move state". $\mathbb{A}(z, x, y)$ thus asserts that, after the (legitimate) configuration z, if Environment does not move, $\mathcal{X}$ reaches the configuration x within y ($y \geq 0$) steps, and it does not move during those steps, either.
- $\mathbb{A}'(z, y)$ abbreviates $\exists x \mathbb{A}(z, x, y)$. $\mathbb{A}'(z, y)$ thus says that, after reaching the (legitimate) configuration z, during the subsequent y (including 0) steps, if Environment does not move, neither does $\mathcal{X}$.

- $\mathbb{B}(z,x)$ says "z is the code of a legitimate configuration C, x is the code of the ith deterministic successor of C for some $i \geq 0$ and, for each j with $0 \leq j \leq i$, the state of the jth deterministic successor of C is a move state iff $j = i$". $\mathbb{B}(z,x)$ thus asserts that z is a legitimate configuration and x is the earliest configuration after (and including) z in which $\mathcal{X}$ moves in the scenario where Environment does not move.

Lemma A.1 **CLA4** $\vdash \sqcup z(z = x \circ y)$.

Proof. Immediate in view of the results of Section 12. ■

In the sequel, whenever we write $\mathfrak{K}$ within a formula or while reasoning in **CLA4**, it is to be understood as a standard variable-free term representing it. For clarity, let us say that this term is 0 followed by $\mathfrak{K}$ $'$s. We shall implicitly rely on the fact that **CLA4** $\vdash \sqcup z(z = \mathfrak{K})$ (Fact 12.6). Similarly for $\mathfrak{k}$, as well as $\ulcorner b \urcorner$ where b is a symbol.

Lemma A.2 **CLA4** $\vdash \sqcup y \mathbb{N}(x,y)$.

Proof. Argue in **CLA4**. By **CLA4**-Induction on z, we first want to show

$$\mathfrak{K} \times |z| \leq |x| \rightarrow \sqcup y\big(|y| \leq |x| \wedge \mathbb{N}([x]_0^{\mathfrak{K} \times |z|}, y)\big). \tag{30}$$

The base $\mathfrak{K} \times |0| \leq |x| \rightarrow \sqcup y\big(|y| \leq |x| \wedge \mathbb{N}([x]_0^{\mathfrak{K} \times |0|}, y)\big)$ is solved by choosing 0 (the code of the empty sequence of symbols) for y. The left inductive step is

$$\Big(\mathfrak{K} \times |z| \leq |x| \rightarrow \sqcup y\big(|y| \leq |x| \wedge \mathbb{N}([x]_0^{\mathfrak{K} \times |z|}, y)\big)\Big) \rightarrow \Big(\mathfrak{K} \times |z\mathbf{0}| \leq |x| \rightarrow \sqcup y\big(|y| \leq |x| \wedge \mathbb{N}([x]_0^{\mathfrak{K} \times |z\mathbf{0}|}, y)\big)\Big).$$

To solve it, we first figure our whether $z = 0$ or $z \neq 0$. If $z = 0$, we ignore the antecedent and do in the consequent the same as what we did in the base case. Otherwise, if $z \neq 0$, we wait till Environment chooses a constant a for y in the antecedent. Then we compute the value b of $[x]^{\mathfrak{K}}_{|z| \times \mathfrak{K}}$ (remember Fact 12.11). If b is $\ulcorner \hat{c} \urcorner$ for some symbol c (which can be established by using Fact 12.8 as many times as the number of symbols), then, using Lemma A.1, we compute $a \circ \ulcorner \check{c} \urcorner$ and choose the computed value for y in the consequent. Otherwise it does not matter what we choose for y, so choose 0. The right inductive step is similar but simpler, as we do not need to give the case $z = 0$ a special consideration. Thus, (30) is proven.

Remember that $\mathfrak{K} = 2^{\mathfrak{k}}$. Let a be the $\mathfrak{k}$th binary predecessor of $|x|$, that is, we have $|x| = a\mathfrak{b}_1 \ldots \mathfrak{b}_{\mathfrak{k}}$, where each $\mathfrak{b}_i$ is either $\mathbf{0}$ or $\mathbf{1}$. Such an a can be found by first computing the value of $|x|$ and then, starting from that value, repeatedly computing binary successor (the constant) $\mathfrak{k}$ times. Let b be the value of the binary predecessor of 2^a. Note that $|b| = a$. Plugging b for z in (30), we make this resource compute a value c for which $\mathbb{N}([x]_0^{\mathfrak{K} \times |b|}, c)$, i.e. $\mathbb{N}([x]_0^{\mathfrak{K} \times a}, c)$, is true. Now notice that, if x is indeed the code of a sequence of symbols, a is the number of symbols in that sequence and, as the length of the code of each symbol is $\mathfrak{K}$, we have $\mathfrak{K} \times a = |x|$; hence, we also have $[x]_0^{\mathfrak{K} \times a} = x$; hence, as $\mathbb{N}([x]_0^{\mathfrak{K} \times a}, c)$ is true, so is $\mathbb{N}(x,c)$. This means that we win the target $\sqcup y \mathbb{N}(x,y)$ by choosing c for y. ■

Lemma A.3 **CLA4** *proves each of the following:*

1. $\mathbb{C}(x) \rightarrow \sqcup y \mathbb{I}(x,y)$.
2. $\mathbb{C}(x) \rightarrow \sqcup y \mathbb{J}(x,y)$.
3. $\mathbb{C}(x) \rightarrow \sqcup y \mathbb{M}(x,y)$.

Proof. Here we only prove clause 1 of the lemma. The remaining clauses are similar.

Argue in **CLA4**. Let $\mathbb{I}'(x,z)$ be (a natural formalization of) the predicate "x is the code of a configuration of the form (29), and $z < i$". By **CLA4**-Induction on z, we want to prove

$$\mathbb{C}(x) \rightarrow \mathbb{I}'(x, |z|) \sqcup \sqcup y(|y| \leq |x| \wedge \mathbb{I}(x,y)\big). \tag{31}$$

The basis is $\mathbb{C}(x) \rightarrow \mathbb{I}'(x, |0|) \sqcup \sqcup y(|y| \leq |x| \wedge \mathbb{I}(x,y)\big)$. We find $[x]^{\mathfrak{K}}_{\mathfrak{K}}$. If the latter is $\underline{\hat{a}}$ for some tape symbol a, we choose the right $\sqcup$-disjunct in the consequent and then choose 0 for y. Otherwise we choose the left disjunct.

The left inductive step is

$$\big(\mathbb{C}(x)\to\mathbb{I}'(x,|z|)\sqcup\sqcup y(|y|\le|x|\wedge\mathbb{I}(x,y)\big)\to\big(\mathbb{C}(x)\to\mathbb{I}'(x,|z\mathbf{0}|)\sqcup\sqcup y(|y|\le|x|\wedge\mathbb{I}(x,y)\big).\tag{32}$$

To solve it, we wait till Environment chooses a $\sqcup$-disjunct in the antecedent. If the left disjunct is chosen, we find $[x]^{\mathfrak{K}}_{|z\mathbf{0}|\times\mathfrak{K}}$. If the latter is $\underline{\hat{a}}$ for some tape symbol a, we choose the right $\sqcup$-disjunct in the consequent of (32), and specify y as the value of $|z\mathbf{0}|$; otherwise we choose the left $\sqcup$-disjunct there. Suppose now the right disjunct is chosen by Environment in the antecedent of (32). We further wait till a constant c is chosen for y there. Then we choose the right $\sqcup$-disjunct in the consequent of (32), and specify y as c in it. The right inductive step is virtually the same. It is not hard to see that our strategy is successful.

Now, the target $\mathbb{C}(x)\to\sqcup y\mathbb{I}(x,y)$ can be seen to be a logical consequence of (31), the **PA**-provable fact $\forall x\forall z\big(z=|x|\to\neg\mathbb{I}'(x,z)\big)$ and the **CLA4**-provable sentence $\sqcap x\sqcup z(z=|x|)$. ■

Lemma A.4 **CLA4** $\vdash\sqcup y\mathbb{E}(x,y)$.

Proof. Argue in **CLA4**. By **CLA4**-Induction on x, we want to show $\sqcup y\big(|y|\le\mathfrak{K}\times|x|\wedge\mathbb{E}(x,y)\big)$, from which the target $\sqcup y\mathbb{E}(x,y)$ immediately follows by LC.

The base $\sqcup y\big(|y|\le\mathfrak{K}\times|0|\wedge\mathbb{E}(0,y)\big)$ is solved by choosing 0 (the code of the empty sequence of symbols) for y. For the left inductive step

$$\sqcup y\big(|y|\le\mathfrak{K}\times|x|\wedge\mathbb{E}(x,y)\big)\to\sqcup y\big(|y|\le\mathfrak{K}\times|x\mathbf{0}|\wedge\mathbb{E}(x\mathbf{0},y)\big),$$

we first figure out whether $x=0$ or not. If yes, we ignore the antecedent and act in the consequent in the same way as in the basis case. Otherwise, we wait till Environment chooses a constant a for y in the antecedent. Then, using Lemma A.1, we compute the value b of $a\circ\ulcorner\check{0}\urcorner$ and choose b for y in the consequent. The right inductive step is similar, with the difference that b should be the value of $a\circ\ulcorner\check{1}\urcorner$ there; also, the case of $x=0$ does not require a special handling. ■

Lemma A.5 **CLA4** $\vdash\neg\exists y\mathbb{D}(x,y)\sqcup\sqcup y\mathbb{D}(x,y)$.

Proof. Argue in **CLA4**. By **CLA4**-Induction on z, we want to show that

$$\mathfrak{K}\times|z|\le|x|\to\neg\exists y\mathbb{D}([x]_0^{\mathfrak{K}\times|z|},y)\sqcup\sqcup y\mathbb{D}([x]_0^{\mathfrak{K}\times|z|},y).\tag{33}$$

The basis $\mathfrak{K}\times|0|\le|x|\to\neg\exists y\mathbb{D}([x]_0^{\mathfrak{K}\times|0|},y)\sqcup\sqcup y\mathbb{D}([x]_0^{\mathfrak{K}\times|0|},y)$ is obviously solved by choosing the right $\sqcup$-disjunct and specifying y as 0 in it.

The left inductive step is

$$\big(\mathfrak{K}\times|z|\le|x|\to\neg\exists y\mathbb{D}([x]_0^{\mathfrak{K}\times|z|},y)\sqcup\sqcup y\mathbb{D}([x]_0^{\mathfrak{K}\times|z|},y)\big)\to\big(\mathfrak{K}\times|z\mathbf{0}|\le|x|\to\neg\exists y\mathbb{D}([x]_0^{\mathfrak{K}\times|z\mathbf{0}|},y)\sqcup\sqcup y\mathbb{D}([x]_0^{\mathfrak{K}\times|z\mathbf{0}|},y)\big).$$

It is solved as follows. If $z=0$, we do in the consequent the same as in the basis case. Suppose now $z\neq0$. We wait till Environment selects a $\sqcup$-disjunct in the antecedent (if there is no such selection, we win). If the left disjunct is selected there, we also select the left disjunct in the consequent and rest our case. Suppose now Environment selects the right disjunct. We wait till it further selects a constant a for y there. Then we compute the value b of $[x]^{\mathfrak{K}}_{\mathfrak{K}\times|z|}$. If $b=\ulcorner\hat{0}\urcorner$, we select the right $\sqcup$-disjunct in the consequent, compute the value c of $a\mathbf{0}$, and specify y as c there. If $b=\ulcorner\hat{1}\urcorner$, we again select the right $\sqcup$-disjunct in the consequent, compute the value c of $a\mathbf{1}$, and specify y as c there. Finally, if neither $b=\ulcorner\hat{0}\urcorner$ nor $b=\ulcorner\hat{1}\urcorner$, we select the left disjunct and retire in victory. The right inductive step is the same, with the only difference that the case $z=0$ does not require a special handling there. Obviously our strategy wins, and (33) is thus proven.

Now, to solve the target $\neg\exists y\mathbb{D}(x,y)\sqcup\sqcup y\mathbb{D}(x,y)$, we use a trick similar to the one employed in the proof of Lemma A.2. Namely, we compute the value a of the $\mathfrak{k}$th binary predecessor of $|x|$, and then the value b of the binary predecessor of 2^a. Note that $|b|=a$. Taking into account that $\mathfrak{K}\times a\le|x|$, by plugging b for z in (33), we essentially turn this resource into

$$\neg\exists y\mathbb{D}([x]_0^{\mathfrak{K}\times a},y)\sqcup\sqcup y\mathbb{D}([x]_0^{\mathfrak{K}\times a},y).$$

This way we would either know that $\neg\exists y\mathbb{D}([x]_0^{\mathfrak{K}\times a}, y)$ is true, or find a constant c for which (we know that) $\mathbb{D}([x]_0^{\mathfrak{K}\times a}, c)$ is true. Note that if x is the code of a sequence of symbols, then $\mathfrak{K}\times a$ is nothing but $|x|$ and hence $[x]_0^{\mathfrak{K}\times a}$ is nothing but x. With this observation in mind, if $\neg\exists y\mathbb{D}([x]_0^{\mathfrak{K}\times a}, y)$ is true, then choosing the left disjunct of $\neg\exists y\mathbb{D}(x, y) \sqcup \sqcup y\mathbb{D}(x, y)$ wins this game; and, if $\mathbb{D}([x]_0^{\mathfrak{K}\times a}, c)$ is true, then choosing the right disjunct and specifying y as c in it wins the game. ■

Lemma A.6 **CLA4** $\vdash \mathbb{C}(x) \to \sqcup y\mathbb{S}(x, y)$.

Proof. Argue in **CLA4** to justify $\mathbb{C}(x) \to \sqcup y\mathbb{S}(x, y)$. Assume that $\mathbb{C}(x)$ is true, namely, that x is (the code of) the configuration (29).

We find the state a of x, which is nothing but $[x]_0^{\mathfrak{K}}$. Next, using clauses 1 and 2 of Lemma A.3, we find the locations i and j of the two scanning heads. Then, using these i and j, we find the symbols b and c seen by the two heads. This allows us to find (within **PA**) the state d of the deterministic successor y of x, the symbol e that will overwrite the old symbol b on the work tape, and the directions in which the heads move. We now correspondingly update x in several steps. First of all, we change the state of x to d. Technically, this is done by computing $\ulcorner d\urcorner \circ [x]_{\mathfrak{K}}^{|x|-\mathfrak{K}}$. In a similar fashion, details of which are left to the reader, we change the old underlined symbol of the work tape to e, and move the two underlines according to the directions in which the corresponding heads move. In addition, if a is a move state, we find the content of the work tape of the original configuration x up to the location of the work-tape head, and append that content, $\top$-prefixed and with each symbol $\hat{\mathfrak{s}}$ changed to $\check{\mathfrak{s}}$ using Lemma A.2, to the content of the run tape of x. Finally, if the previously blank cell $\#m$ of the work tape is no longer blank, we insert a blank cell to the right of it in our representation of the configuration (again, technical details about how, exactly, all this can be done, are left as an easy exercise for the reader). The eventual value of x, after the above updates, will be exactly the sought value of the deterministic successor of the original x, that is, the value that we should choose for y in the consequent of $\mathbb{C}(x) \to \sqcup y\mathbb{S}(x, y)$. ■

Lemma A.7 **CLA4** $\vdash \mathbb{C}(z) \to \mathbb{A}'(z, |r|) \sqcup \sqcup x\mathbb{B}(z, x)$.

Proof. Argue in **CLA4**. By **CLA4**-Induction on r, we want to prove

$$\mathbb{C}(z) \to \sqcup x\big(|x| \leq |z| + |r| \wedge \mathbb{A}(z, x, |r|)\big) \sqcup \sqcup x\big(|x| \leq (|z| + |r|)\mathbf{0} \wedge \mathbb{B}(z, x)\big),$$

from which the target $\mathbb{C}(z) \to \mathbb{A}'(z, |r|) \sqcup \sqcup x\mathbb{B}(z, x)$ easily follows by LC.

To solve the base $\mathbb{C}(z) \to \sqcup x\big(|x| \leq |z| + |0| \wedge \mathbb{A}(z, x, |0|)\big) \sqcup \sqcup x\big(|x| \leq (|z| + |0|)\mathbf{0} \wedge \mathbb{B}(z, x)\big)$, we figure out whether the state of z is a move state or not. If yes, we choose the right $\sqcup$-disjunct; if not, we choose the left $\sqcup$-disjunct. In either case, we further choose the value of z for the variable x and win.

The left inductive step is

$$\begin{array}{l}\Big(\mathbb{C}(z) \to \sqcup x\big(|x| \leq |z| + |r| \wedge \mathbb{A}(z, x, |r|)\big) \sqcup \sqcup x\big(|x| \leq (|z| + |r|)\mathbf{0} \wedge \mathbb{B}(z, x)\big)\Big) \;\to \\ \Big(\mathbb{C}(z) \to \sqcup x\big(|x| \leq |z| + |r\mathbf{0}| \wedge \mathbb{A}(z, x, |r\mathbf{0}|)\big) \sqcup \sqcup x\big(|x| \leq (|z| + |r\mathbf{0}|)\mathbf{0} \wedge \mathbb{B}(z, x)\big)\Big).\end{array} \tag{34}$$

If $r = 0$, (34) is won by solving its consequent in the same ways as the basis case was solved. Suppose now $r \neq 0$. To solve (34), we wait till Environment selects one of the two $\sqcup$-disjuncts in the antecedent.

If the right $\sqcup$-disjunct is selected, we wait further till a constant c for x is selected there. Then we select the right $\sqcup$-disjunct in the consequent, and choose the same c for x in it.

Suppose now the left $\sqcup$-disjunct is selected in the antecedent of (34). Wait further till a constant c for x is selected there. We may assume that $\mathbb{A}(z, c, |r|)$ is true, or else we win the game. Using Lemma A.6, we find the deterministic successor d of the configuration c. With a little thought, one can see that the size of d cannot exceed the sum of the sizes of z and $r\mathbf{0}$ more than twice, so that $|d| \leq (|z| + |r\mathbf{0}|)\mathbf{0}$ holds. We figure out whether the state of d is a move state or not. If not, we select the left $\sqcup$-disjunct in the consequent of (34), otherwise, select the right disjunct. In either case, we further choose d for x and win.

The right inductive step is virtually the same, with the only difference that the case $z = 0$ does not require a special handling. ■

A.2 Proof of clause (a) of Lemma 14.6

Assume the conditions of clause (a) of Lemma 14.6.

First, let us consider the case where $H_i(y,\vec{s})$ is the result of replacing in $E(\vec{s})$ a surface occurrence of a subformula $F_0 \sqcap F_1$ by F_j (j=0 or j=1). Let $\bot\alpha$ be the labmove that brings $E(\vec{s})$ down to $H_i(y,\vec{s})$. For instance, if $E(\vec{s})$ is $G \to F_0 \sqcap F_1$ and $H_i(y,\vec{s})$ is $G \to F_0$, then $\bot\alpha$ is "$\bot$1.0".

Argue in **CLA4** to justify $E^\circ_\circ(z,\vec{s}) \to \sqcup u H^\circ_i(u,y,\vec{s})$. Assume $E^\circ_\circ(z,\vec{s})$. This implies that z is a legitimate configuration with yield $E(\vec{s})$, and that the same holds for the deterministic successor b of z, which we compute using Lemma A.6. Then the result of appending $\bot\alpha$ to the run tape contents of (the configuration encoded by) b is a legitimate configuration with yield $H_i(y,\vec{s})$. The code of such a configuration is $b \circ \ulcorner \check{\bot}\check{\alpha} \urcorner$. We compute the value c of the latter using Lemma A.1, and win $E^\circ_\circ(z,\vec{s}) \to \sqcup u H^\circ_i(u,y,\vec{s})$ by choosing c for u.

Next, consider the case where $H_i(y,\vec{s})$ is the result of replacing in $E(\vec{s})$ a surface occurrence of a subformula $\sqcap x F(x)$ by $F(y)$. Let α be the string such that, for any constant c, the labmove $\bot\alpha c$ brings $E(\vec{s})$ down to $H_i(c,\vec{s})$. For instance, if $E(\vec{s})$ is $G \to \sqcap x F(x) \sqcup J$ and $H_i(y,\vec{s})$ is $G \to F(y) \sqcup J$, then α is "1.0."; and if $E(\vec{s})$ is just $\sqcap x F(x)$, then α is the empty string.

Argue in **CLA4** to justify $E^\circ_\circ(z,\vec{s}) \to \sqcup u H^\circ_i(u,y,\vec{s})$. Let c be the value satisfying $\mathbb{E}(y,c)$. We compute the latter using Lemma A.4. Assume $E^\circ_\circ(z,\vec{s})$. This implies that z is a legitimate configuration with yield $E(\vec{s})$. Then the same holds for the deterministic successor b of z, which we compute using Lemma A.6. Then the result of appending $\bot\alpha y$ to the run tape contents of (the configuration encoded by) b is a legitimate configuration with yield $H(y,\vec{s})$. The code of such a configuration is $b \circ \ulcorner \check{\bot}\check{\alpha} \urcorner \circ c$. We compute the value d of the latter using Lemma A.1, and win $E^\circ_\circ(z,\vec{s}) \to \sqcup u H^\circ_i(u,y,\vec{s})$ by choosing d for u.

A.3 Proof of clause (b) of Lemma 14.6

Assume the conditions of clause (b) of Lemma 14.6. In **CLA4**, we can solve (25) as follows. Assume $E^\circ(z,\vec{s})$, which, of course, implies $\mathbb{C}(z)$. We compute the value of the binary predecessor of $2^{\chi(|z|)}$, and use that value to specify r in the resource of Lemma A.7. As a result, we get the resource $\mathbb{A}'\big(z,\chi(|z|)\big) \sqcup \sqcup x \mathbb{B}(z,x)$. This means that we will either know that $\mathbb{A}'\big(z,\chi(|z|)\big)$ is true, or find a constant a for which we will know that $\mathbb{B}(z,a)$ is true.

If $\mathbb{A}'\big(z,\chi(|z|)\big)$ is true, then so is $E^\circ_\circ(z,\vec{s})$ and, by choosing the latter, we win (25).

Now, for the rest of this proof, suppose $\mathbb{B}(z,a)$ is true. Using clause 3 of Lemma A.3, we find the number i with $\mathbb{I}(a,i)$. Then we find (the code of) the move α that $\mathcal{X}$ made in a. Namely, $\ulcorner \hat{\alpha} \urcorner = [a]^{\mathfrak{K}\times i}_{\mathfrak{K}}$. Using Lemma A.6, we also find the deterministic successor b of a. Fix these α and b.

Let $\beta_1,\ldots,\beta_m$ be all legal moves in position $E(\vec{s})$ that signify a choice of one of the two $\sqcup$-disjuncts in some surface subformula $F_0 \sqcup F_1$ of $E(\vec{s})$. Let $H_1(y,\vec{s}),\ldots,H_m(y,\vec{s})$ be the corresponding $(\bot,y)$-developments of $E(\vec{s})$.

Further, let $\beta_{m+1},\ldots,\beta_n$ be all strings such that, any move signifying a choice of a constant c for x in some surface subformula $\sqcap x F(x)$ of $E(\vec{s})$ looks like $\beta_i c$ for some $i \in \{m{+}1,\ldots,n\}$. Let $H_{m+1}(y,\vec{s}),\ldots,H_n(y,\vec{s})$ be the corresponding $(\bot,y)$-developments of $E(\vec{s})$.

To solve (25), we need to solve its consequent, which now can be rewritten as follows:

$$E^\circ_\circ(z,\vec{s}) \sqcup \mathbb{L} \sqcup \sqcup u \sqcup y H^\circ_1(u,y,\vec{s}) \sqcup \ldots \sqcup \sqcup u \sqcup y H^\circ_m(u,y,\vec{s}) \sqcup \sqcup u \sqcup y H^\circ_{m+1}(u,y,\vec{s}) \sqcup \ldots \sqcup \sqcup u \sqcup y H^\circ_n(u,y,\vec{s}). \tag{35}$$

This is how we solve (35). First, for each $i \in \{1,\ldots,m\}$, we compare $\ulcorner \hat{\alpha} \urcorner$ with $\ulcorner \hat{\beta}_i \urcorner$. **If** they turn out to be the same, **then** we choose the disjunct $\sqcup u \sqcup y H^\circ_i(u,y,\vec{s})$ in (35), and specify u and y as b and 0 in it, respectively. Here our choice of 0 for y is arbitrary and has no effect on the game, as $H^\circ_i(u,y,\vec{s})$ does not contain the variable y, anyway.

Otherwise, for each $i \in \{m{+}1,\ldots,n\}$, we compare $[\ulcorner \hat{\alpha} \urcorner]^{r_i}_0$ with $\ulcorner \hat{\beta}_i \urcorner$, where r_i is the size of $\ulcorner \hat{\beta}_i \urcorner$. **If** they turn out to be the same, **then**, we figure out the (code of the) "rest" γ of the string α. That is, γ is the string such that $\alpha = \beta_i\gamma$. Employing Lemma A.5, we either find a number c with $\mathbb{D}(\ulcorner \hat{\gamma} \urcorner, c)$, or ($\sqcup$) find out that such a c does not exist ($\neg\exists$). In the former case, we choose the disjunct $\sqcup u \sqcup y H^\circ_i(u,y,\vec{s})$ in (35) and specify u and y as b and c in it, respectively. In the latter case, α is an illegal move, so we choose $\mathbb{L}$, which is true because $\mathcal{X}$, having made an illegal move, loses.

Otherwise, α is simply an illegal move, so we (again) choose $\mathbb{L}$.
It is left to the reader to convince himself or herself that our strategy succeeds.

References

[1] S. Abramsky and R. Jagadeesan. *Games and full completeness for multiplicative linear logic*. **Journal of Symbolic Logic** 59 (1994), pp. 543-574.

[2] K. Aehlig, U. Berger, M. Hoffmann and H. Schwichtenberg. *An arithmetic for non-size-increasing polynomial-time computation*. **Theoretical Computer Science** 318 (2004), pp. 3-27.

[3] S. Bellantoni and M. Hoffmann. *A new "feasible" arithmetic*. **Journal of Symbolic Logic** 67 (2002), pp. 104-116.

[4] A. Blass. *Degrees of indeterminacy of games*. **Fundamenta Mathematicae** 77 (1972), pp. 151-166.

[5] A. Blass. *A game semantics for linear logic*. **Annals of Pure and Applied Logic** 56 (1992), pp. 183-220.

[6] S. Buss. **Bounded Arithmetic** (revised version of Ph. D. thesis). Bibliopolis, 1986.

[7] S. Buss. *The polynomial hierarchy and intuitionistic bounded arithmetic*. **Lecture Notes in Computer Science** 223 (1986), pp. 77-103.

[8] P. Clote and G. Takeuti. *Bounded arithmetic for NC, ALogTIME, L and NL*. **Annals of Pure and Applied Logic** 56 (1992), pp. 73-117.

[9] J. Girard. *Light linear logic*. **Information and Computation** 143 (1998), pp. 175-204.

[10] J. Girard, A. Scedrov and P. Scott. *Bounded linear logic: a modular approach to polynomial-time computability*. **Theoretical Computer Science** 97 (1992), pp. 1-66.

[11] P. Hajek and P. Pudlak. **Metamathematics of First-Order Arithmetic**. Springer, 1993.

[12] G. Japaridze. *The logic of tasks*. **Annals of Pure and Applied Logic** 117 (2002), pp. 263-295.

[13] G. Japaridze. *Introduction to computability logic*. **Annals of Pure and Applied Logic** 123 (2003), pp. 1-99.

[14] G. Japaridze. *Propositional computability logic I*. **ACM Transactions on Computational Logic** 7 (2006), pp. 302-330.

[15] G. Japaridze. *Propositional computability logic II*. **ACM Transactions on Computational Logic** 7 (2006), pp. 331-362.

[16] G. Japaridze. *Introduction to cirquent calculus and abstract resource semantics*. **Journal of Logic and Computation** 16 (2006), pp. 489-532.

[17] G. Japaridze. *Computability logic: a formal theory of interaction*. In: **Interactive Computation: The New Paradigm**. D. Goldin, S. Smolka and P. Wegner, eds. Springer 2006, pp. 183-223.

[18] G. Japaridze. *From truth to computability I*. **Theoretical Computer Science** 357 (2006), pp. 100-135.

[19] G. Japaridze. *From truth to computability II*. **Theoretical Computer Science** 379 (2007), pp. 20-52.

[20] G. Japaridze. *The logic of interactive Turing reduction*. **Journal of Symbolic Logic** 72 (2007), pp. 243-276.

[21] G. Japaridze. *Intuitionistic computability logic*. **Acta Cybernetica** 18 (2007), pp. 77-113.

[22] G. Japaridze. *The intuitionistic fragment of computability logic at the propositional level*. **Annals of Pure and Applied Logic** 147 (2007), pp.187-227.

[23] G. Japaridze. *Cirquent calculus deepened*. **Journal of Logic and Computation** 18 (2008), pp. 983-1028.

[24] G. Japaridze. *Sequential operators in computability logic*. **Information and Computation** 206 (2008), pp. 1443-1475.

[25] G. Japaridze. *Many concepts and two logics of algorithmic reduction*. **Studia Logica** 91 (2009), pp. 1-24.

[26] G. Japaridze. *In the beginning was game semantics*. **Games: Unifying Logic, Language, and Philosophy**. O. Majer, A.-V. Pietarinen and T. Tulenheimo, eds. Springer 2009, pp. 249-350.

[27] G. Japaridze. *Towards applied theories based on computability logic*. **Journal of Symbolic Logic** 75 (2010), pp. 565-601.

[28] G. Japaridze. *Toggling operators in computability logic*. **Theoretical Computer Science** 412 (2011), pp. 971-1004.

[29] G. Japaridze. *A logical basis for constructive systems*. **Journal of Logic and Computation** (in press). doi: 10.1093/logcom/exr009, 38 pages.

[30] G. Japaridze. *Introduction to clarithmetic II*. Manuscript at http://arxiv.org/abs/1004.3236

[31] G. Japaridze. *Introduction to clarithmetic III*. Manuscript at http://arxiv.org/abs/1008.0770

[32] J. Krajicek. **Bounded Arithmetic, Propositional Logic, and Complexity Theory**. Cambridge University Press, 1995.

[33] D. Leivant. *Intrinsic theories and computational complexity*. **Lecture Notes in Computer Science** 960 (1995), pp. 117-194.

[34] I. Mezhirov and N. Vereshchagin. *On abstract resource semantics and computability logic*. **Journal of Computer and System Sciences** 76 (2010), pp. 356-372.

[35] H. Schwichtenberg. *An arithmetic for polynomial-time computation*. **Theoretical Computer Science** 357 (2006), pp. 202-214.

[36] W. Xu and S. Liu. *Knowledge representation and reasoning based on computability logic*. **Journal of Jilin University** 47 (2009), pp. 1230-1236.